②

FINAL TECHNICAL REPORT TO

MANPOWER RESEARCH AND DEVELOPMENT PROGRAM

by

Jeffery L. Kennington and Richard V. Helgason
Department of Computer Science and Engineering
Southern Methodist University
Dallas, TX 75275-0122
(214)-692-3278 and (214)-768-3079

for

High Speed Heuristics For Real-Time Personnel Assignment Models

26 January 1994

N00014-92-J-1619

SMU # 5-25151

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Unrestricted
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION SMU — 6b. OFFICE SYMBOL (If applicable) CSE	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code) Dallas, TX 75275-0122	7b. ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Office of Naval Research — 8b. OFFICE SYMBOL (If applicable) ONR	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code) 800 North Quincy Street Arlington, VA 22217-5000	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
"High Speed Heuristics for Real-Time Personnel Assignment Models"

12. PERSONAL AUTHOR(S) Jeffery L. Kennington and Richard V. Helgason

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final	FROM 12/91 TO 12/93	94, 1, 26	

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This document presents a new network based model (called the cloning model) for the problem of on-line personnel assignment. In computer simulation tests, we found that the specialized software designed to solve the cloning model will obtain optimal solutions in about four seconds on a 486 PC running at 50 Mhz. This demonstrates that this model can be used for on-line applications of personnel assignment which involve telephone negotiation. In addition, this document presents new algorithms for a variety of optimization models including (i) the singly constrained assignment problem, (ii) the separable convex cost network flow problem, (iii) the minimum cost network flow problem, and (iv) the problem of identifying the extreme points of the convex hull of a given set of points.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified	
22a. NAME OF RESPONSIBLE INDIVIDUAL Jeffery L. Kennington	22b. TELEPHONE (Include Area Code) (214) 768-3278	22c. OFFICE SYMBOL CSE

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
Unclassified

Table of Contents

I. STATEMENT OF WORK

Optimization algorithms and their implementation in software provide the computational engines needed to develop new and improved computer systems for enlisted personnel assignment. Specifically, network based optimization models are frequently used to model problems in the personnel assignment area. The current trend is to develop on-line (as opposed to batch) computer systems to assist detailers in making job assignments. This means that iterative optimization algorithms that could run in batch mode and take thirty minutes of computer time on an IBM mainframe, are now expected to obtain an optimal solution in only a few seconds on a personal computer. This requires new approaches to both the models used and the algorithms implemented. In this report, we present new ideas in both of these arenas.

II. PUBLICATIONS

Title

On-Line Algorithms for Navy Enlisted Personnel Assignment

Author

Jeffery L. Kennington

Executive Summary

This manuscript presents a pair of on-line optimization models for use by assignment detailers during telephone negotiation with enlisted personnel. The decision problem in the environment in which job assignments are made on-line without allowing an enlisted man to have any choice may be modeled as an on-line bipartite matching problem. Using an empirical analysis, the theoretical best algorithm for this model was shown to also work very well in practice. The decision problem in the environment in which an enlisted man is given a choice from among a fixed number of candidate assignments may be modeled as a minimum cost flow problem. In empirical tests on both a Dec Alpha workstation and a 486 based PC, it is shown that the specialized software for this model is sufficiently fast that it can be used in on-line applications.

Publication Status

This paper has been submitted for publication and is currently under review.

Title

The Singly Constrained Assignment Problem: A Lagrangean Relaxation Heuristic Algorithm

Authors

Jeffery L. Kennington and Farin Mohammadi

Executive Summary

Many Navy personnel assignment problems can be modeled as some version of an assignment problem or an assignment problem with side constraints. This manuscript presents a heuristic method for finding near optimal integer assignments for the singly constrained assignment problem. Lagrangean duality theory is used to develop a robust procedure which can easily obtain integer solutions guaranteed to be within 1% of an optimum for problems having up to 50,000 binary variables. In empirical tests with software implementations of all competing techniques, our software was found to be superior in terms of both quality of solution and solution time. We believe that this is the current best algorithm and software implementation for this class of problems.

Publication Status

This paper has been accepted for publication in Computational Optimization and Applications.

Title

Solution of Convex Cost Network Flow Problems Via Linear Approximation

Authors

Richard V. Helgason and Rajluxmi V. Murthy

Executive Summary

This paper presents a specialized algorithm for solving the separable convex cost network flow problem. The method involves solving a piece–wise linear approximation which yields an upper bound for the original problem. A lower bound is obtained using either the Frank–Wolfe method or a Lagrangean method. If the two bounds satisfy a termination criteria, then the procedure terminates; otherwise, the piece–wise linear approximation is refined near the current solution and the procedure is repeated. In an empirical evaluation with quadratic problems, a software implementation of the new algorithm was found to be about four times faster than a comparable software implementation of the Frank–Wolfe method.

Publication Status

This paper has been accepted for publication in Optimization Methods and Software.

Title

Computational Study of Implementational Strategies of the Network Penalty Method

Authors

Nandagopal Venugopal and Richard V. Helgason

Executive Summary

This paper presents a pair of algorithms for the minimal cost network flow problem based on the network penalty method of Conn, Gamble, and Pulleyblank. In an empirical analysis, a software implementation of the primal network simplex method was found to be superior to software implementations of the network penalty method.

Publication Status

This paper has not been submitted for publication.

Title

A Nearly Asynchronous Parallel LP-based Algorithm for the Convex Hull Problem in Multidimensional Space

Authors

Jose Dula, Richard V. Helgason, and Nandagopal Venugopal

Executive Summary

The frame problem is to find the extreme points of the convex hull of a given set of points. Alternately, one wishes to classify the points of a given set into those which are extreme points of the convex hull of the given set and those which are not. Applications of the frame problem appear in stochastic programming, data envelopment analysis, and redundancy determination in linear programming. This paper presents a new parallel algorithm for this problem along with an empirical analysis of the algorithm. Speedups of seven were obtained using fourteen processors on a twenty processor Sequent Symmetry S81.

Publication Status

This paper has been submitted for publication and is currently under review.

Technical Report 93-CSE-30

On-Line Algorithms for Navy Enlisted Personnel Assignment

By

Jeffery L. Kennington
(214) smu-3278
jlk@seas.smu.edu

Department of Computer Science and Engineering
Southern Methodist University
Dallas, TX 75275-0122

July 1993

Abstract

Each year over 200,000 Navy enlisted personnel are assigned new jobs at a cost of over 250 million dollars in moving expenses. The personnel involved in assignment prefer an on-line system which allows for telephone negotiation between the enlisted person and his detailer who will make the new assignment. This manuscript presents a pair of mathematical models and algorithms which can be used for on-line assignment. One model assumes that the assignment decision rests solely with the detailer who makes decisions which are simply relayed to the caller. This model is known in the literature as on-line bipartite matching. The second model assumes that both the enlisted man and the detailer play a role in selecting the new assignment. The detailer prepares a list of potential jobs from which the enlisted man may choose. The underlying mathematical model is a network program. The feasibility of using network optimization for this on-line system is demonstrated in a simulation study.

Acknowledgment

This research was supported by the Navy Personnel Research and Development Center and the Office of Naval Research under Contract Number N00014-91-J-1234.

I. INTRODUCTION

During the last eight years, analysts at the Navy Personnel Research and Development Center have developed and installed two completely different personnel assignment systems. These systems were designed to assist some three hundred detailers at the Navy Annex in Washington D. C. with the task of assigning 200,000 Navy personnel to new jobs each year. The Enlisted Personnel Allocation and Nomination System (EPANS) solves an optimization problem off-line in a batch operation. The Computer Enhanced Detailing and Distribution System (CEDAD) is an on-line system that produces a list of recommended jobs for a given person. While EPANS provides near optimal assignments based on the given eligibility rules, it was not warmly embraced by the detailers for whom it was designed. The on-line CEDAD system has been a great success and will be the model for future production systems in the area of personnel assignment.

The CEDAD on-line system is similar to an airline reservation system and is used to assist a detailer during telephone negotiation with Navy personnel who are up for reassignment. A man who is a candidate for reassignment calls his detailer and requests information about open positions. While on the phone, the detailer uses the CEDAD system to produce a list of potential assignments for this caller. The list of potential assignments is a function of the skills of the caller, the availability of a technical school which may be required for a particular assignment, timing issues regarding when a ship is scheduled to embark, and the technical requirements of the jobs. Based on the list produced by CEDAD, the detailer suggests a few possibilities and the caller and the detailer negotiate a new assignment. Once an assignment has been agreed upon, the detailer initiates the process to generate orders for this caller and this job is removed from the system. This entire process is usually handled in a single telephone conversation. At the end of the month, all men who have not been assigned are matched to jobs by the detailer. Each month begins a new cycle and a typical problem involves 200 men. Unassigned jobs are carried forward to the next month and some jobs remain vacant. Most ships put to sea with less than the full authorized crew.

In this study we present two mathematical models related to Navy personnel assignment along with simulation results obtained from applying these models. A binary eligibility matrix B where $b_{ij} = 1$ implies that man i is eligible for job j is input for both models. The eligibility rules are not documented and are the subject of much debate within the Navy. Different dispatchers (who are the experts in Navy personnel assignment) will produce different eligibility matrices for the same set of men and jobs. For this analysis, it is assumed that a unique eligibility matrix can be developed.

II. ON-LINE ASSIGNMENT WITHOUT CHOICE

Traditionally the Navy has attempted to allow its 600,000 people to have some influence over their assignments. The telephone numbers of the 300 dispatchers are published and it is well-known that an individual can talk to his dispatcher about their new assignment. The dispatchers are also in the Navy are given instructions about how assignments should be made. For the case in which the dispatcher makes the decision and relays the new assignment to the caller, there exists a mathematical model which has appeared in the literature under the title on-line bipartite matching.

The key feature of the on-line bipartite matching model is that the dispatcher only obtains access to the rows of the eligibility matrix as the men arrive for service (assignment). This is exactly the information provided to the dispatcher by *the CEDAD system. In the context of the problem of interest, the on-line bipartite* matching problem may be stated as follows:

> **Given n men and n jobs, suppose the men arrive at random and request an assignment. Upon the arrival of man i, the ith row of the eligibility matrix is presented to the dispatcher along with a list of previously assigned jobs. From the available assignments (if one exists) the dispatcher selects a job and the assignment for man i is made. The objective of the dispatcher is to maximize the total number of assignments.**

The input for this problem is the eligibility matrix which is revealed to the dispatcher one row at a time.

If a particular man is only eligible for a few jobs, this would not be known to the dispatcher until man i arrived for service. Suppose the current eligibility matrix is given by

$$B = \begin{bmatrix} 1111 \\ 1111 \\ 1100 \\ 1100 \end{bmatrix}$$

and the men arrive in the order 1, 2, 3, 4. If the dispatcher assigned man 1 to job 1 and man 2 to job 2, then neither of the remaining men could be assigned. Hence, only 2 assignments would be made when 4 are possible. If the dispatcher was presented B prior to the first arrival, then he could solve the bipartite matching problem off-line and simply reveal the optimal assignments as the requests are made.

Theoretical results and an optimal algorithm for the on-line bipartite matching problem have been presented by Karp, Vazirani, and Vazirani [1990]. For this type of model, a worst-case analysis compares the performance of a given algorithm against an adaptive on-line adversary. For the bipartite matching problem, this means that the adversary gets to specify row i+1 of the eligibility matrix, B, after assignments have been made for the first i men. The adversary must also construct his own perfect matching using the same eligibility matrix presented to the algorithm. This can be viewed as a zero-sum two person game in which one of the players is selecting the algorithm to be applied and the other player, called the adversary, selects the input data (the eligibility matrix) in an attempt to foil the first players algorithm. In this way a worst-case analysis can be developed.

The simplest algorithm for this problem is known as a greedy algorithm. As each man is presented, the corresponding row of the eligibility matrix is scanned and the first unassigned job is selected. This may be described as follows:

Greedy On-Line Algorithm

Inputs:	n	- denotes the number of men and jobs
	b[i,j]	- equals 1 if man i is eligible for job j, and 0; otherwise
Output:	man[i]	- equals j if job j is assigned to man i, and 0; otherwise

```
begin
   $J \leftarrow \Phi$;
   for i=1,...,n do
      man[i] $\leftarrow$ 0, E $\leftarrow$ $\{j:b[i,j]=1\}\backslash J$;
      if E $\neq \Phi$, then j $\leftarrow$ min(k:k $\in$ E), man[i] $\leftarrow$ j, J $\leftarrow J \cup \{j\}$;
   end for
end
```

An adversary can produce an eligibility matrix for which there is a perfect matching (every man is assigned to a job) in which the greedy algorithm will only achieve n/2 assignments. Consider the following algorithm:

Worst Adversary for the Greedy Algorithm

Input:	n	- denotes the number of men and jobs
Outputs:	greedyman[i]	- equals j if man i is assigned to job j by the greedy algorithm, and 0; otherwise
	advman[i]	- equals the job assigned to man i by the adversary
Assumption:	n is even	

begin

```
  p ← n/2;
  for i=1,...,p do
    for j=1,...,n do b[i,j] ← 1, greedyman[i] ← i, advman[i] ← n+1-i;
  end for
  for i=p+1,...,n do
    greedyman[i] ← 0;
    for j=1,...n do b[i,i-p] ← 1, advman[i] ← i-p;
  end for
end
```

One approach which has been used to foil an adversary is to introduce ramdomization into the algorithm. Since the adversary can not forecast the precise assignments made by the algorithm, a randomized algorithm is superior to a greedy algorithm. This method may be stated as follows:

Randomized On-Line Algorithm

Inputs:	n	- denotes the number of men and jobs
	b[i,j]	- equals 1 if man i is eligible for job j, and 0; otherwise
Output:	man[i]	- equals j if job j is assigned to man i, and 0; otherwise

```
begin
  J ← Φ;
  for i=1,...,n do
    man[i] ← 0, E ← {j:b[i,j]=1}\J;
    if $E \neq \Phi$, then select j randomly from E, man[i] $\leftarrow$ j, J $\leftarrow$ J $\cup$ {j};
  end for
end
```

The following is an adversary for the randomized algorithm:

Adversary for the Randomized On-Line Algorithm

Input:	n	- denotes the number of men and jobs
Outputs:	randomman[i]	- equals j if man i is assigned to job j by the randomized algorithm, and 0; otherwise
	advman[i]	- equals the job assigned to man i by the adversary
Assumption:	n is even	

```
begin
  p $\leftarrow$ n/2, R $\leftarrow$ $\Phi$, A $\leftarrow$ $\Phi$;
  for i=1,...,p do
    for j=1,...,n do b[i,j] $\leftarrow$ 1;
    for all j $\in$ R $\cup$ A do b[i,j] $\leftarrow$ 0;
    E $\leftarrow$ {j:b[i,j]=1};
    select r at random from E\R, randomman[i] $\leftarrow$ r, R $\leftarrow$ R $\cup$ {r};
    select a at random from E\A,    advman[i] $\leftarrow$ a, A $\leftarrow$ A $\cup$ {a};
  end for
  for i=p+1,...,n do
    randomman[i] $\leftarrow$0;
    for j=1,...,n do b[i,j] $\leftarrow$ 1;
    for all j $\in$ A do b[i,j] $\leftarrow$ 0;
```

```
        E ← {j:b[i,j]=1};
        select a at random from E, advman[i] ← a, A ← A ∪ {a};
        if E\R ≠ Φ, select r at random from E\R, randomman[i] ← r, R ← R ∪ {r};
    end for
end
```

Karp, Vazirani, and Vazirani [1990] indicate that the randomized algorithm performs poorly on the following matrix:

$$b_{ij} = \begin{cases} 1, \text{ if } i = j \text{ or } 1 \leq i \leq n/2 \text{ and } n/2 \leq j \leq n \\ 0, \text{ otherwise} \end{cases}$$

where n is even. They claim that the randomized algorithm gives too high a priority to the jobs {n/2,...,n} during the first n/2 assignments. An algorithm known as the ranking method, given below, corrects this fallacy.

The Ranking Algorithm

Inputs:	n	- denotes the number of men and jobs
	b[i,j]	- equals 1 if man i is eligible for job j, and 0; otherwise
Output:	man[i]	- equals j if job j is assigned to man i, and 0; otherwise
Other:	index[p]	- denotes the job with the pth priority

```
begin
    E ← {1,...,n};
    for p=1,...,n do select j at random from E, index[p] ← j, E ← E\{j};
    J ← Φ;
```

```
  for i=1,...,n do
    stop ← 'no', p ← 1;
    while stop = 'no' and p ≤ n do
      j ← index[p];
      if j ∉ J and b[i,j]=1 then
        man[i] ← j, J ← J ∪ {j}, stop ← 'yes';
      else
        p ← p + 1;
      end if
    end while
  end for
end
```

Karp, Vazirani, and Vazirani prove that the on-line ranking algorithm achieves the best possible performance for this problem.

It is well-known that the theoretically best algorithm may not be the best practical procedure. The inadequacy of the worst-case criterion for judging algorithms is clearly demonstrated in the comparison of the simplex method with the ellipsoid method for solving linear programs. In a worst-case analysis, the ellipsoid method is superior to the simplex method, but to my knowledge all empirical analyses have shown that the simplex method is far superior to the ellipsoid method for practical application. The folklore about the ellipsoid method is that the observed behavior is similar to the worst-case bound whereas the observed behavior of the simplex method is much better than the worst-case bound. An excellent discussion of both algorithms may be found in Chvatal [1983].

The three algorithms have been implemented in software and tested on 150 randomly generated test problems. For each problem density, fifteen problems were solved using each of the three algorithms. The number of assignments achieved by each algorithm was recorded in three vectors of length fifteen. These vectors were sorted and the minimum, median, and maximum values achieved are presented in Table 1. The test problems were generated so that a perfect matching exists. For the 5% dense problems, the best matching obtained by the *greedy algorithm* was 391 men whereas both *randomized* and *ranking* achieved a best matching of 392 men.

============================

Table 1 About Here

============================

The important results are illustrated graphically in Figures 1 and 2. The upper end of each bar represents the best matching achieved, the lower end represents the worst achieved, and the rectangle indicates the median for the fifteen problems. For the low density problems illustrated in Figure 1, the *ranking algorithm* is superior. As the density increased (see Figure 2), there was essentially no difference among the three algorithms.

============================

Figures 1 and 2 About Here

============================

The best possible outcome in the analysis of an algorithm is for it to have the best theoretical bound and be the best performer in an empirical analysis. Our simulation results show that the *ranking algorithm* for on-line bipartite matching is not only best using the worst-case criteria, but is also best in an empirical analysis. A weaknesses of this empirical analysis is that it involved random problems. Hence, applying a greedy algorithm to a random problem may be viewed as the same as the ranking algorithm. This helps explain why the greedy algorithm works so well in the empirical analysis. For Navy enlisted personnel assignment in which choice is not allowed, this study clearly indicates that the *ranking algorithm* is the preferred method.

III. ON-LINE ASSIGNMENT WITH CHOICE

In the Navy it is generally believed that allowing personnel to influence their assignments has a positive effect on morale. However, everyone can not be assigned to their first choice and some individuals must rotate through the less desirable assignments. For the model presented in this section, individuals will be given a list of potential assignments from which they can choose. The size of the list will be controlled and the jobs placed on the list will be carefully selected by an optimization model.

For this model we assume that the entire eligibility matrix is available to the dispatcher prior to the first arrival. We also assume that a cost, c_{ij} is incurred when man i is assigned to job j. These costs are related to Navy policies which are used to guide dispatchers in making assignments. Under these assumptions, the problem of on-line assignment with choice from a list of size k can be defined as follows:

> **Given n men and m>n jobs, suppose the men arrive at random and request an assignment. Upon the arrival of man i, the dispatcher generates a list of potential assignments. The list must have k jobs unless man i is eligible for fewer than k unassigned jobs in which case all possible assignments are presented. Man i chooses his next assignment from this list. Assuming that an unassigned man must pay a high cost, the objective of the dispatcher is to achieve a small total cost for all assignments.**

The inputs for this problem are the eligibility matrix, B, the cost matrix, C, and the list size, k.

For the special case in which k=1, the problem could be solved off-line as a network model and the assignments could be revealed as the men arrive. The network structure for this model is illustrated in Figure 3. The nodes m1, ..., m4 represent the four men each with a supply of 1, and the nodes j1, ..., j6 represent the six jobs. The sink has a demand of 4 which will absorb all of the supply. All arcs from man i to job j have a unit cost of c_{ij}, a lower bound of 0, and an upper bound of 1. All arcs from man i to the dummy job have a unit cost of 0, a lower bound of 0, and no upper bound. The arcs from job j to the sink have unit cost and lower bound of 0 and upper bound of 1. The arc from the dummy job to the sink has infinite unit cost, infinite upper bound, and 0 lower bound. Solving this model will produce an optimal assignment for the special case in which k=1.

For the case in which k>1, we propose a similar network model in which a given man is cloned k times. For the model illustrated in Figure 3, suppose man 3 arrives for service and k=3. The corresponding network model is illustrated in Figure 4. Note that only two requirements were modified to obtain the Figure 4 model from the Figure 3 model. The supply for man 3 is 3 and the demand at the sink is 6. Suppose the flows given in (.) are optimal for the Figure 4 model. Then man 3 would be allowed to choose his assignment from the list consisting of jobs 3 and 4. Suppose man 3 chooses job 4 and man 1 arrives next. The new model is illustrated in Figure 5. Note that only three changes were made to the model in Figure 4 to obtain the model in Figure 5. The supply of man 1 was changed from 1 to k=3, the supply of man 3 (previous arrival) was changed from 3 to 1, and the lower bound on the arc from man 3 to job 4 was set to 1 (i. e. man 3 is assigned to job 4).

The reason we show the modifications required to change the model in Figure 3 to the model in Figure 4 and the model in Figure 4 to the model in Figure 5 is because we envision a system that uses the optimal solution to one model as an advanced starting solution to obtain the optimal solution to the next model. For a 200 man problem, we would solve the model in Figure 3 off-line before the first man arrived. When the first man arrived, we would clone that man and solve the corresponding network problem. The solution would be presented and the man would choose his new assignment. When the next man arrives, the process would be repeated.

A simulation system has been developed to obtain information on both the quality of the solutions which could be expected and the computer time required to solve the network models. The network models were solved with a special version of the network solver MODFLO (see Ali and Kennington [1989]). A summary of the results are given in Table 2. All the models had 602 nodes and from 10,000 to 20,000 arcs. The time to solve the base model illustrated in Figure 4 appears under the column entitled Optimum Using MODFLO. These times varied from a low of 2 seconds to a high of 7 seconds. All times are wall clock times on a Dec Alpha machine which has a retail price of approximately $100,000. For each of the fifteen problems, a simulation was run in which the men arrive randomly. Hence, to simulate one month with a 200 man problem requires solving 201 network problems. Each of these runs was replicated 5 times to give a grand total of 15,015 solved problems to accumulate the data in Table 2.

===================================

Table 2 About Here

===================================

For the k=4 problems having 100 jobs/man or 20,000 arcs, the first problem was solved in 3.8 seconds, and the 1000 other problems were solved in less than 0.3 seconds each. That is, the reoptimization never took more than 10% of the time required to obtain the first solution. The median times for all cases was much less than 1 second.

The Navy has expressed an interest in developing a distributed system in which a dispatcher makes his calculations on a PC after downloading the data from the mainframe. Table 3 gives a summary of the same information run on a 486 based PC running at 50 Mhz. The median times for reoptimization are acceptable for a PC based system.

==================================

Table 3 About Here

==================================

The data in Tables 2 and 3 also present the trade-off between increased values of k (the number of jobs presented to a caller) and the resulting objective value. The median values are displayed in Figure 6. If four jobs are offered to each caller, one can expect an overall assignment to be about 2.7 times worse than the best possible. When ten jobs are offered, the objective value is about 6 times worse.

==================================

Figure 6 About Here

==================================

IV. SUMMARY AND CONCLUSIONS

This manuscript presents two on-line models and corresponding algorithms for the assignment of Navy enlisted personnel. The case in which reassignment is made without allowing an enlisted man to have any choice can be modeled as an on-line bipartite matching problem. The theoretical best on-line bipartite matching algorithm was also shown to be best in an empirical study. The case in which reassignment is made with an enlisted man having a choice from among k potential assignments has been modeled as a pure network problem. The feasibility of this approach was demonstrated in a simulation model. By using specialized software designed to solve network problems and using the previous solution as an advanced start for the new optimization problem, the solution times on a 486 based PC were only a few seconds. We also observed that a large list size results in solutions which have a large deviation from an optimal solution produced using a list size of 1.

REFERENCES

Ali, A., and J. Kennington, [1989], "MODFLO User's Guide," Technical Report 89-OR-03, Southern Methodist University, Dallas, TX 75275.

Chvatal, V., [1983], Linear Programming, W. H. Freeman and Company, NY, NY.

Karp, R., U. Vazirani, and V. Vazirani, [1990], "An Optimal Algorithm for On-Line Bipartite Matching," STOC: Proceedings of the 22nd ACM Symposium on Theory of Computing, 352-358.

Table 1. Empirical Analysis of On–Line Algorithms for Bipartite Matching (For a given density, 15 400x400 bipartite matching problems were solved using each of the algorithms)

Eligibility Matrix		On–Line Algorithms		
Size nxn	Density	Greedy (min,median,max)	Randomized (min,median,max)	Ranking (min,median,max)
400	5%	383, 387, 391	383, 387, 392	384, 388, 392
400	10%	392, 394, 396	391, 394, 396	392, 395, 397
400	15%	394, 396, 398	392, 396, 398	395, 396, 398
400	20%	396, 398, 398	394, 398, 399	395, 397, 399
400	25%	396, 398, 399	396, 398, 400	395, 398, 400
400	30%	397, 399, 400	395, 398, 399	397, 398, 400
400	35%	397, 399, 399	396, 398, 400	397, 399, 400
400	40%	399, 399, 399	397, 399, 400	398, 399, 400
400	45%	399, 399, 399	396, 399, 400	397, 399, 400
400	50%	399, 399, 399	398, 399, 400	398, 399, 400

Table 2. Simulation Results from On-Line Cioning Model
(All times are on a Dec Alpha)

Jobs Offered To Each Caller K	Elgibility Matrix		Optimum Using MODFLO		Simulation Results					
	Size n x m	Number Jobs Per Man	Scaled Obj	Time (secs)	Scaled Obj Value (min, median, max)			Time in seconds (min,median,max)		
4	200 x 400	50	100	3.94	274	281	288	0.02	0.08	0.36
4	200 x 400	75	100	3.00	284	296	322	0.03	0.08	0.26
4	200 x 400	100	100	3.80	274	278	282	0.03	0.10	0.29
6	200 x 400	50	100	2.07	387	409	434	0.03	0.11	1.19
6	200 x 400	75	100	4.13	361	394	425	0.03	0.15	1.09
6	200 x 400	100	100	3.91	414	433	477	0.06	0.16	0.32
8	200 x 400	50	100	2.03	486	509	513	0.04	0.10	0.25
8	200 x 400	75	100	3.04	540	566	577	0.06	0.15	0.30
8	200 x 400	100	100	3.83	530	561	581	0.08	0.21	1.26
10	200 x 400	50	100	4.07	654	703	729	0.06	0.24	0.77
10	200 x 400	75	100	5.91	624	632	668	0.08	0.20	0.82
10	200 x 400	100	100	3.76	619	630	657	0.12	0.27	7.80
12	200 x 400	50	100	4.48	725	847	866	0.09	0.33	6.48
12	200 x 400	75	100	5.43	866	909	952	0.12	0.27	1.22
12	200 x 400	100	100	7.04	721	759	787	0.15	0.55	1.47

Table 3. Simulation Results from On-Line Clone Model
(All times are on a 486 PC running at 50 Mhz)

Jobs Offered To Each Caller	Elgibility Matrix		Optimum Using MODFLO		Simulation Results					
	Size m x n	Number Jobs Per Man	Scaled Obj	Time (secs)	Scaled Obj Value (min, median, max)			Time in seconds (min,median,max)		
4	200 x 400	40	100	29.22	258	280	294	0.38	1.20	2.58
4	200 x 400	60	100	42.02	293	301	313	0.61	1.70	4.56
4	200 x 400	80	100	54.93	270	294	309	0.88	2.30	6.81
6	200 x 400	40	100	29.27	392	400	427	0.76	1.76	3.95
6	200 x 400	60	100	44.27	408	443	489	1.04	2.52	5.60
6	200 x 400	80	100	54.99	373	392	397	1.26	3.41	7.14
8	200 x 400	40	100	30.21	518	544	619	0.93	2.25	4.45
8	200 x 400	60	100	42.78	469	502	510	1.26	3.35	51.19
8	200 x 400	80	100	55.37	461	519	526	2.93	4.45	10.06
10	200 x 400	40	100	28.40	566	624	651	1.21	2.91	36.20
10	200 x 400	60	100	42.73	627	650	670	1.76	3.90	7.64
10	200 x 400	80	100	55.92	701	721	745	2.19	5.33	11.10
12	200 x 400	40	100	30.15	751	787	814	3.29	18.89	41.80
12	200 x 400	60	100	42.08	751	767	820	2.03	4.61	8.85
12	200 x 400	80	100	57.17	733	815	831	2.80	6.37	13.02

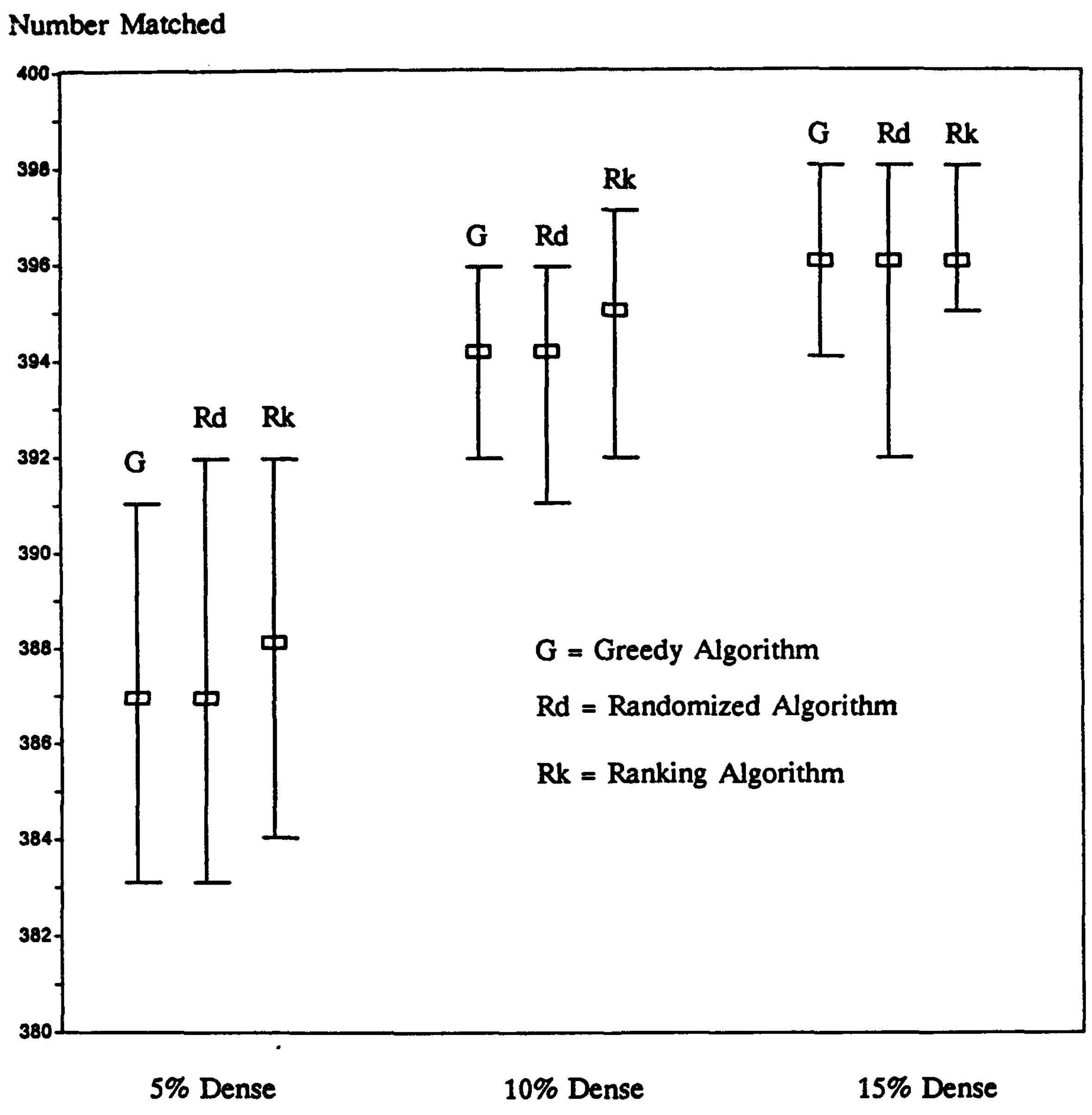


Figure 1. Performance of On-Line Matching Algorithms for 400x400 Bipartite Matching Problems Having an Arc Density of at Most 15%

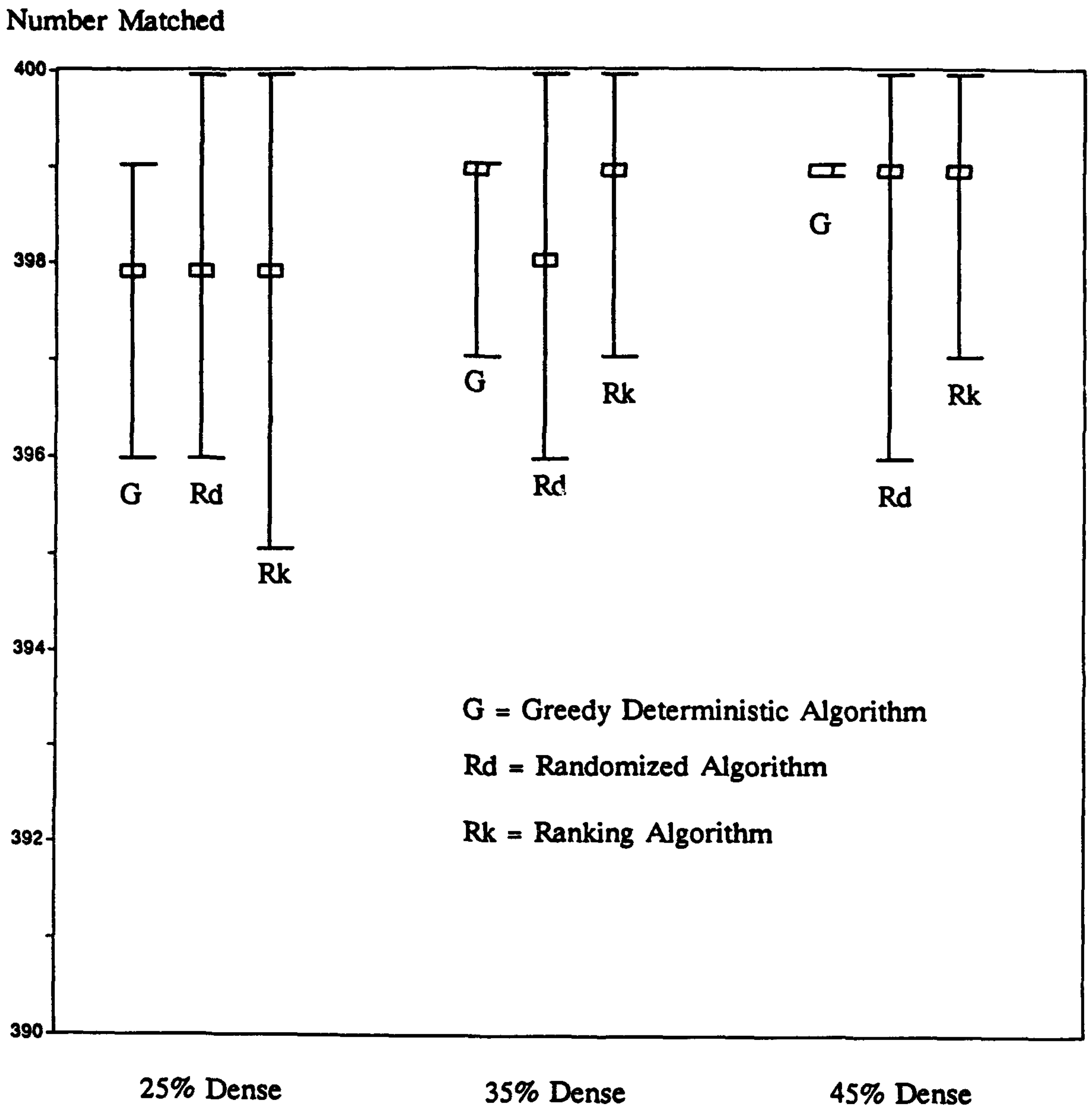


Figure 2. Performance of On-Line Matching Algorithms for 400x400 Bipartite Matching Problems Having an Arc Density of at Least 25%

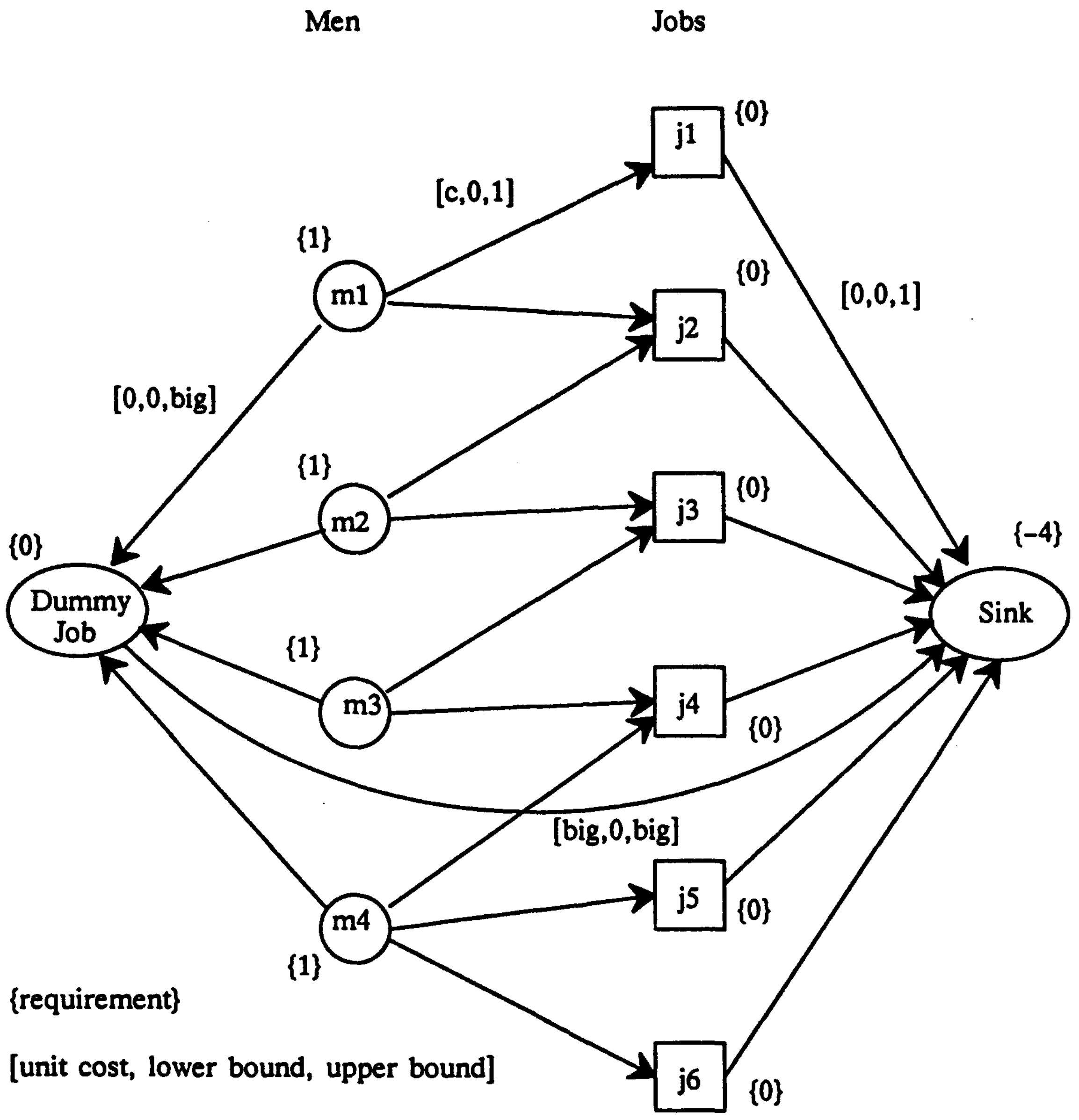


Figure 3. Network Model for Off-Line Assignment (4 men and 6 jobs)

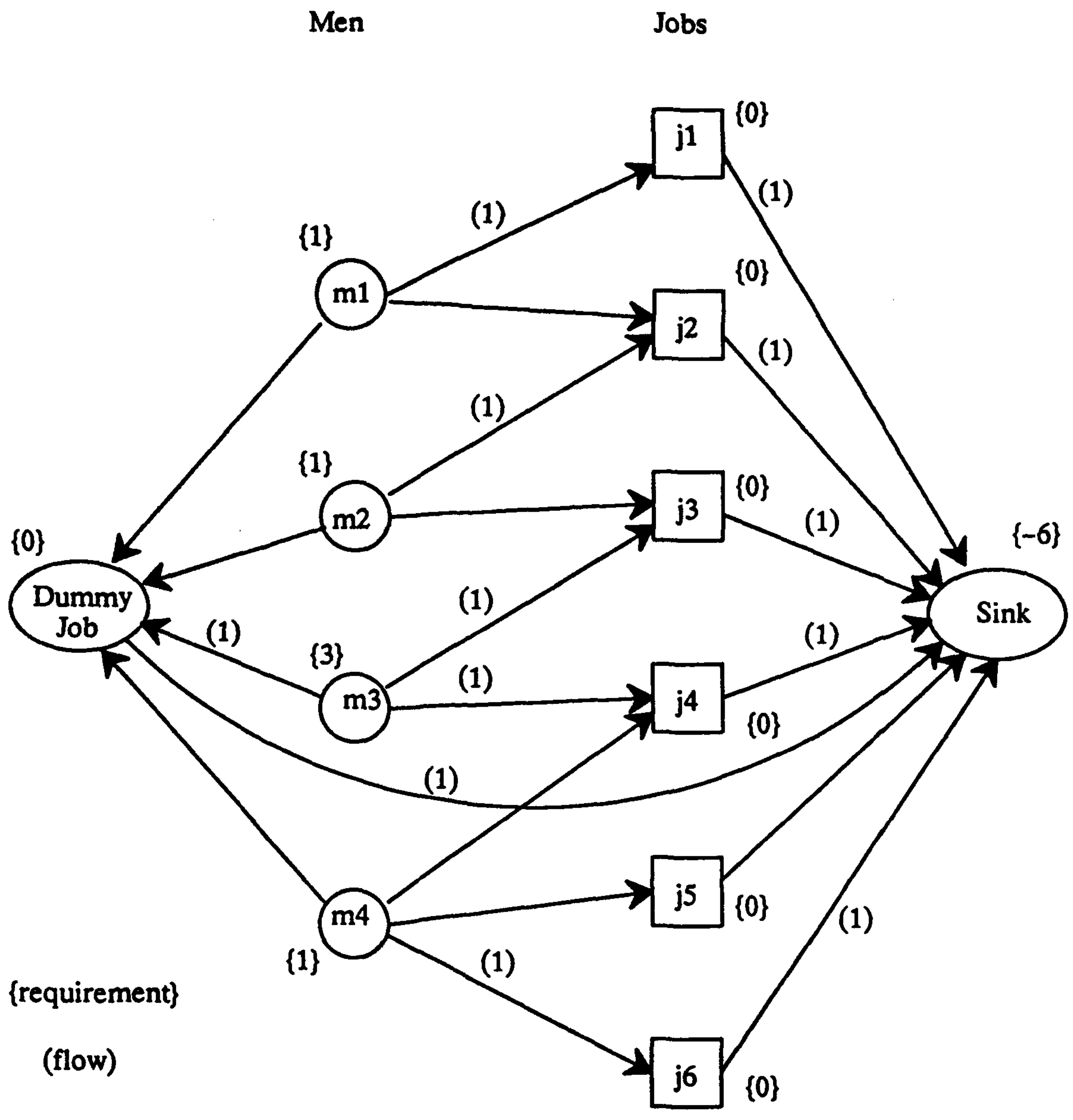


Figure 4. Network Model for On-Line Assignment with Three Choices

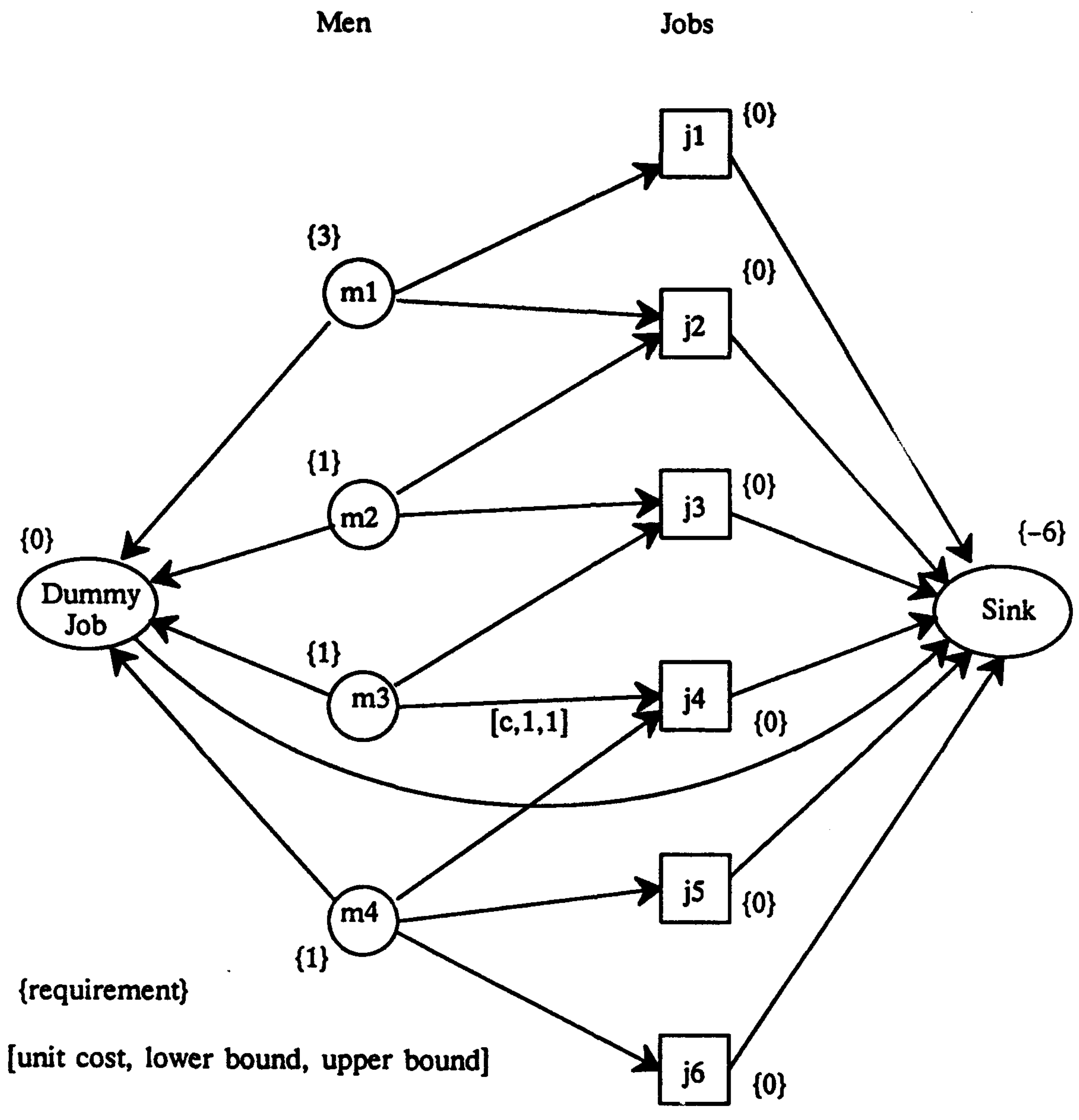


Figure 5. Network Model when Man 1 Arrives (Man 3 is Assigned to Job 4)

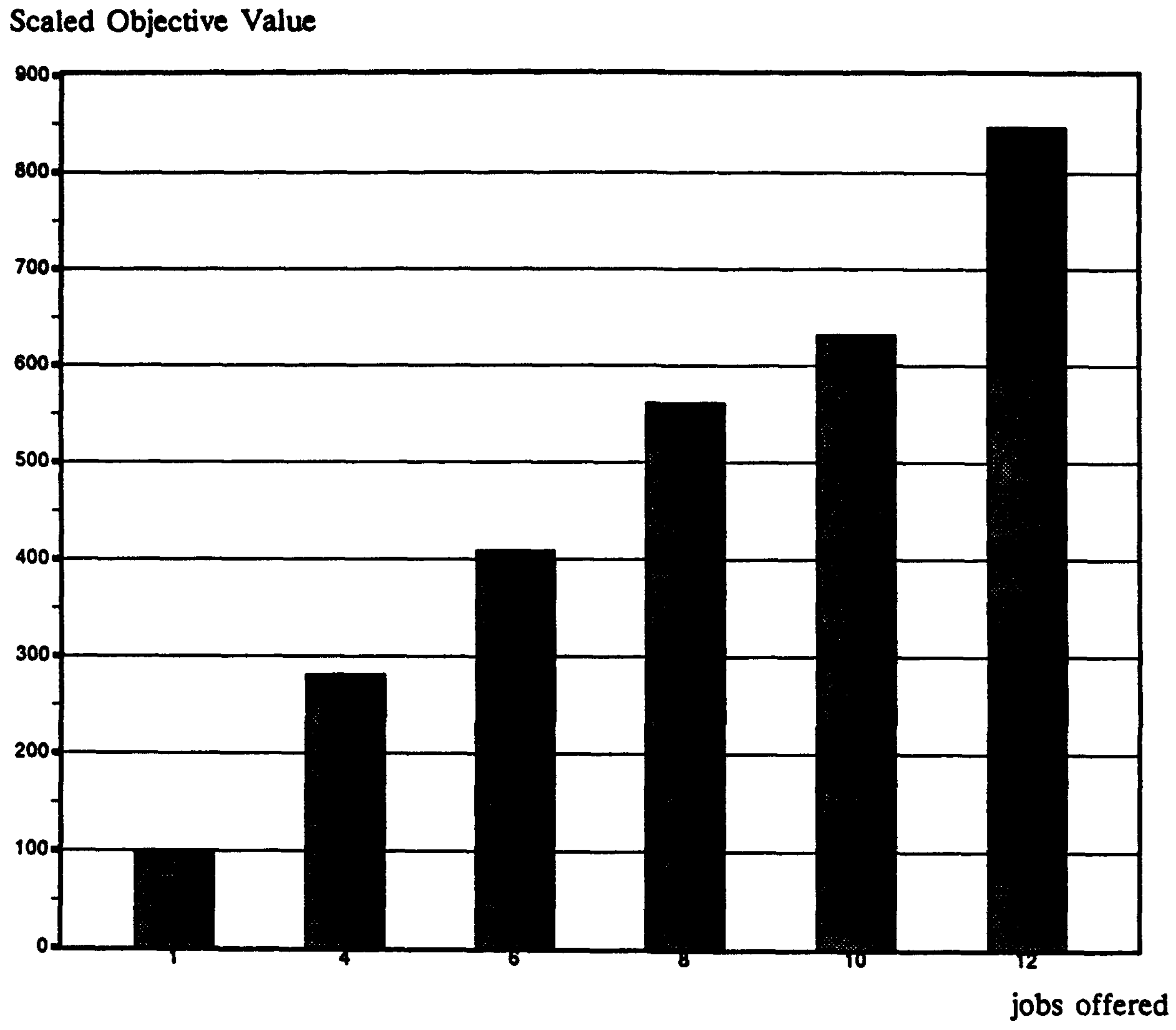


Figure 6. The Scaled Objective Value As a Function of the Number of Jobs Offered to a Caller

Technical Report 91-CSE-1

THE SINGLY CONSTRAINED ASSIGNMENT PROBLEM: A LAGRANGEAN RELAXATION HEURISTIC ALGORITHM

Jeffery L. Kennington
(214)-768-3278
jlk@seas.smu.edu
&
Farin Mohammadi
(214)-768-1476
fam@seas.smu.edu

Department of Computer Science and Engineering
School of Engineering and Applied Science
Southern Methodist University
Dallas, Texas 75275-0122

revised June 1993

Comments and criticisms from interested readers are cordially invited.

ABSTRACT

This manuscript presents a new heuristic algorithm to find near optimal integer solutions for the singly constrained assignment problem. The method is based on Lagrangean duality theory and involves solving a series of pure assignment problems. The software implementation of this heuristic, ASSIGN+1, successfully solved problems having one-half million binary variables (assignment arcs) in less than seventeen minutes of wall clock time on a Sequent Symmetry S81 using a single processor. In computational comparisons with MPSX and OSL on an IBM 3081D, the specialized software was from one hundred to one thousand times faster. In computational comparisons with the specialized code of Mazzola and Neebe, we found that ASSIGN+1 was forty times faster. In computational comparisons with our best alternating path specialized code, we found that ASSIGN+1 was more than three times faster than that code. This new software proved to be very robust as well as fast. The robustness is due to an elaborate scheme used to update the Lagrangean multipliers and the speed is due to the fine code used to solve the pure assignment problems. We also present a modification of the algorithm for the case in which the number of jobs exceeds the number of men along with an empirical analysis of the modified software.

ACKNOWLEDGMENT

This research was supported in part by the Air Force Office of Scientific Research under Contract Number AFSOR 91-0028, and the Office of Naval Research under Contract Number N00014-87-K-0223.

I. INTRODUCTION

The singly constrained assignment problem is to determine a least cost assignment of n men to n jobs such that a single additional constraint is satisfied. This model is a special case of a binary linear program and may be stated mathematically as follows:

$$\text{minimize} \quad \sum_{(i,j) \in E} c_{ij} x_{ij} \tag{1}$$

$$\text{subject to} \quad \sum_{j:(i,j) \in E} x_{ij} = 1 , \quad i = 1, \ldots, n \tag{2}$$

$$\sum_{i:(i,j) \in E} x_{ij} = 1 , \quad j = 1, \ldots, n \tag{3}$$

$$x_{ij} \in \{0, 1\} , \quad \text{all } (i,j) \in E \tag{4}$$

$$\sum_{(i,j) \in E} a_{ij} x_{ij} \leq b \tag{5}$$

where c_{ij} denotes the cost for assigning man i to job j, a_{ij} denotes the coefficient of x_{ij} in the side constraint, b denotes the right-hand-side of the side constraint, E is the set of (man, job) pairs corresponding to eligible assignments, and $x_{ij} = 1$ implies that man i is assigned to job j. In order to simplify the notation we let **a** denote the vector corresponding to the coefficients in (5), x denote the vector corresponding to the binary decision variables, c denote the vector of costs, and T= { x : (2), (3), and (4) }. Then the singly constrained assignment problem can be stated as $P_1 = \min \{cx : x \in T \text{ and } ax \leq b\}$, and it is well-known that P_1 is NP-complete.

The singly constrained assignment model was first used by Brans, Leclercq, and Hansen [6] to model the core management of a pressurized water reactor. The problem is: given two sets of fresh and exposed assemblies determine the location pattern of these assemblies which maximizes the reactivity of the core under a constraint on power-distribution form factor. After linearization, Brans, Leclercq, and Hansen reduce the problem to a sequence of singly constrained assignment problems and propose an implicit enumeration routine to solve these problems. Our work on P_1 was motivated by models which had been developed by analysts at the Navy Personnel Research and Development Center in San Diego. These models involve the optimal assignment of men to jobs under a budget constraint related to relocation cost.

The first specialized algorithm for P_1 was presented by Gupta and Sharma [14]. Their method was a straight forward enumeration scheme and they present no computational results. Aggarwal [1] presents an improved algorithm for P_1 which combines Lagrangean-relaxation with the enumeration algorithm of Gupta and Sharma [14] to obtain an optimal solution. No computational results for this method is presented. Mazzola and Neebe [23] developed a two phase algorithm for the constrained assignment problem. Phase I is a subgradient optimization based heuristic procedure that obtains near optimal integer solutions and phase II is a branch-and-bound procedure which obtains the optimal solution. To generate solutions at each node of the branch-and-bound tree they developed a method that combines a restricted basis pivoting rule followed by a subgradient routine. Their empirical evaluation of the heuristic and the branch-and-bound algorithm indicates that both procedures are satisfactory for dense assignment problems of size up to 100x100 and the phase I procedure obtains near optimal integer solutions for most problems tested. Bryson [7] presents an algorithm based on the parametric programming procedure of Gass and Saaty [9]. The largest problem they solved had fewer than 2000 edges and did not exploit the network structure of this model. This is in contrast to our empirical investigation in which the small problems have over a quarter of a million edges. Ball, Derigs, Hilbrand, and Metz [3] present an algorithm which will solve the special case of P_1 in which $a_{ij} \in \{0,1\}$ for all (i,j).

The work by Klingman and Russell [20] and Barr, Farhangian, and Kennington [5] is for the continuous version of P_1 rather than the binary version. That is, the above work would be applicable for the model in which (4) is replaced with the nonnegativity constraint $x_{ij} \geq 0$, all $(i,j) \in E$.

Klingman and Russell [21] developed a simplex based method for the transportation problem with a single side constraint and Glover, Karney, Klingman, and Russell [13] developed a simplex based method for the transshipment problem with a single side constraint. Authors of both papers state that codes based on their procedures are significantly faster than the LP code APEX–III and they both obtain an integer solution for the problem with an inequality side constraint by pivoting into the basis the slack variable associated with the side constraint. This yields a triangular basis which automatically produces an integer solution. Empirically these integer solutions were found to be within 1% of optimality.

An extension of the alternating path basis algorithm of Barr, Glover, and Klingman [4] for the singly constrained assignment problem may be found in Kennington and Mohammadi [19]. Integer feasible solutions are then obtained by a forced pivot with the slack variable associated with the side constraint. This results in a lower triangular basis and a corresponding integer feasible solution. This solution is not guaranteed to be optimal, but empirical analysis has shown that solutions obtained this way are quite good.

Since (1)-(5) is a binary linear program, all the literature on integer programming applies (see Everett [8], Geoffrion [10, 11], Glover [12], Salkin [28], Shapiro [29, 30], Parker and Rardin [27], Nemhauser and Wolsey [24]). In practice most integer programming models are either solved as a linear program and the solutions are rounded using some heuristic or branch-and-bound is used in an attempt to obtain a solution within a prespecified tolerance.

The objective of this study is to present a new algorithm for the singly constrained assignment problem. The algorithm is for the problem having an inequality side constraint. We also show how this algorithm can be used to solve problems in which (5) is an equality and problems in which (3) is replaced with

$$\sum_{i\,:\,(i,j)\,\in\,E} x_{ij} \leq 1 \;,\; j = 1, \ldots, m. \tag{6}$$

The algorithm uses a Lagrangean relaxation and solves a series of assignment problems. Empirical results demonstrate the superiority of this approach over competing software. Problems having one-half million arcs were solved in less than fifteen minutes on a Sequent Symmetry S81 using a single processor.

II. THE ALGORITHM

In this section we present a heuristic algorithm for the singly constrained assignment problem, $P_1 = \min \{ cx : x \in T, ax \leq b \}$. Let $P_2 = \min \{ cx : x \in T \}$ be a feasible region relaxation of P_1, and let $P_3 = \min \{ ax-b : x \in T \}$. By dualizing the side constraint one obtains a Lagrangean relaxation of P_1 given by $P(\beta) = \min \{ cx + \beta(ax-b) : x \in T \}$ where β is the Lagrangean multiplier. Let $v[P]$ denote the optimal objective function value for any problem P, then a Lagrangean dual for P_1 is $LD_1 = \max \{ v[P(\beta)] : \beta \geq 0 \}$.

We attempt to solve the Lagrangean dual, LD_1, by solving the problems P_2, P_3 and a series of $P(\beta)$ for different values of β. P_2 is solved to obtain the initial lower bound, *lb*, and to determine if the side constraint is redundant. The solution to P_3 either establishes that P_1 has no feasible solution or provides an initial upper bound, *ub*. Solving the Lagrangean relaxation, $P(\beta)$, always provides a lower bound and if the optimal solution, x_β, is feasible for P_1, then cx_β is an upper bound.

It is well known that $v[P(\beta)]$ is a piece-wise linear concave function over R^+. Let x_β denote an optimum for $P(\beta)$ at any point β. Let β^* denote an optimum for LD_1. LD_1 may have a unique optimum as illustrated in Figure 1 or may have an infinite number of solutions as illustrated in Figure 2. For the case illustrated in Figure 1, for all $\beta > \beta^*$, $ax_\beta < b$ and x_β is feasible for P_1, and for all $\beta < \beta^*$, $ax_\beta > b$ and x_β is not feasible for P_1. For the case illustrated in Figure 2, for all $\beta > \beta^*$, either $ax_\beta < b$ and x_β is feasible for P_1 or $ax_\beta = b$ and x_β is an optimum for P_1. Similarly for all $\beta < \beta^*$, either $ax_\beta > b$ and x_β is not feasible for P_1 or $ax_\beta = b$ and x_β is an optimum for P_1.

Figures 1 and 2 here

Let u and v denote upper and lower bounds, respectively on β^*. Then a bisection search can be used to obtain β^*. Since obtaining each value of $v[P(\beta)]$ for a givenβ requires solving an assignment problem, we attempt to obtain a small interval ofuncertainty $[u,v]$ prior to initiating the bisection search. It is well-known that if theduality gap is zero, then $v[LD_1]=v[P_1]$. That is,

$$cx_{\beta^*} + \beta^*(ax_{\beta^*} - b) = v[P_1] \text{ , or} \tag{7}$$

$$\beta^* = (v[P_1] - cx_{\beta^*})/(ax_{\beta^*} - b). \tag{8}$$

Prior to using the bisection search, we obtain estimates for β^* using (8) with $v[P_1]$ replaced by $(lb+ub)/2$, cx_{β^*} replaced by ub, and ax_{β^*} replaced by ax_3. That is, the initial value of β is given by:

$$\beta = (ub - lb)/(2(ax_3 - b)). \tag{9}$$

In the bisection search we may apply the normal bisection ($\beta=(u+v)/2$) or the slope bisection as described by Ryu and Guignard [26]. The bisection technique using the the slopes is a special case of the intersection algorithm of Yu [31]. The new β is selected such that $cx_u + \beta(ax_u - b) = cx_v + \beta(ax_v - b)$. That is, $\beta=(cx_u - cx_v)/(ax_v - ax_u)$. This selection is illustrated in Figure 3. Note that if $ax_v - b = s_3$ and $ax_u - b = s_4$, then the new β will be β^*. Ryu and Guignard [26] prove that this technique *produces an optimal* β *in a finite number of* iterations.

Figure 3 here

Using these ideas, the basic strategy for the new heuristic algorithm for P_1 may be described as follows:

```
step 1.  find an initial lower bound,
step 2.  find an initial upper bound,
while $\mathbf{ax}_\beta < b$ and stopping criteria not satisfied repeat steps 3 and 4,
    step 3.  use (9) to estimate $\beta$ then solve $P(\beta)$,
    step 4.  update $u$, $v$, $ub$, and $lb$ if possible,
while stopping criteria is not satisfied repeat steps 5-7,
    step 5.  update $\beta$ using normal or slope bisection and solve $P(\beta)$,
    step 6.  if $\mathbf{ax}_\beta < b$ update $u$, $ub$, and if possible $lb$,
    step 7.  if $\mathbf{ax}_\beta > b$ update $v$ and if possible $ub$ and $lb$.
```

When the slope bisection search is used, the algorithm is terminated when either $v[P(\beta)] = \mathbf{cx}_u + \beta(\mathbf{ax}_u - b)$ or $v[P(\beta)] = \mathbf{cx}_v + \beta(\mathbf{ax}_v - b)$. For the normal bisection search, termination occurs when either $|u-v| \leq \text{tol}$ or $ub-lb \leq .01lb$.

The ASSIGN+1 algorithm for P_1 is described below:

Input:

1. The cost vector, **c**.
2. The side constraint, vector **a** and constant b.
3. The set of (man, job) pairs corresponding to eligible assignments, E.
4. For a slope bisection line search, initialize *search* to slope bisection; otherwise, initialize *search* to normal bisection.
5. Termination tolerance, tol.

Output:

1. The solution vector, y.
2. A lower bound for P_1, *lb*.
3. The objective value corresponding to y, *ub*.
4. The termination status. If P_1 has no feasible solution, then

```
        *status* = infeasible; if an optimal solution was obtained then
        *status*=optimal; otherwise, *status* = feasible.
algorithm ASSIGN+1:
begin
    FIND INITIAL LOWER BOUND;
    FIND INITIAL UPPER BOUND;
    FIND INITIAL U;
    Run:= go;
    while $\gamma<0$ do REDUCE U;
    while Run=go do BISECTION;
    if $u>\beta$, then $\beta := u$ and let $x_\beta$ be an optimum for $P(\beta)$;
    $y := x_\beta$;
end
procedure FIND INITIAL LOWER BOUND
begin
    let $x_2$ be an optimum for $P_2$. $v := 0$. $lb := cx_2$;
    if $ax_2 \le b$. then status:= optimal, $y := x_2$. $ub := cy$, stop;
end
procedure FIND INITIAL UPPER BOUND
begin
    let $x_3$ be an optimum for $P_3$, $\delta := ax_3 - b$;
    if $\delta>0$, then status:= infeasible, stop;
    else status:= feasible, $ub := cx_3$, $\gamma := 1$, $\beta := (lb-ub)/(2\delta)$;
end
```

```
Procedure FIND INITIAL U
begin
    let $\mathbf{x}_\beta$ be an optimum for $P(\beta)$;
    $\gamma := \mathbf{a}\mathbf{x}_\beta - b$, $lb := \max\{ lb, v[P(\beta)] \}$;
    if $\gamma = 0$, then $status :=$ optimal, $y := \mathbf{x}_\beta$, $ub := \mathbf{c}y$, stop;
    if $\gamma < 0$, then $ub := \min\{ ub, \mathbf{c}\mathbf{x}_\beta \}$, $u := \beta$;
    else $v := \max\{ v, \beta \}$ , $\beta := 2\beta$, FIND INITIAL U;
end
procedure REDUCE U
begin
    if $\beta = (lb-ub)/(2\delta)$, then $\beta := \beta/2$;
    else $\beta := (lb-ub)/(2\delta)$;
    let $\mathbf{x}_\beta$ be an optimum for $P(\beta)$;
    $\gamma := \mathbf{a}\mathbf{x}_\beta - b$, $lb := \max\{ lb, v[P(\beta)] \}$;
    if $\gamma = 0$, then $status :=$ optimal, $y := \mathbf{x}_\beta$, $ub := \mathbf{c}y$, stop;
    if $\gamma < 0$, then $ub := \min\{ ub, \mathbf{c}\mathbf{x}_\beta \}$, $u := \min\{ u, \beta \}$;
    else $v := \beta$;
end
procedure BISECTION
begin
    if $search =$ slope bisection, then $\beta := (\mathbf{c}\mathbf{x}_u - \mathbf{c}\mathbf{x}_v)/(\mathbf{a}\mathbf{x}_v - \mathbf{a}\mathbf{x}_u)$;
    else $\beta := (u+v)/2$;
    let $\mathbf{x}_\beta$ be an optimum for $P(\beta)$;
```

```
        $\gamma$:= $ax_\beta$ -b, *lb*:= max{ *lb*,v[P($\beta$)] };
        **if** $\gamma$=0, **then** *status*:= optimal, y:= $x_\beta$, *ub*:= cy, and stop;
        **if** $\gamma$<0, **then** *ub*:= min{ *ub*, $cx_\beta$ }, and *u*:= min{ *u*,$\beta$ };
        **else** *v*:= max{ *v*,$\beta$ };
        **if** *search*=normal bisection, **then if** |*u*-*v*|<tol or |*ub*-*lb*|<.01*lb*, **then**
          *Run*:= stop;
        **else**
          **if** v[P($\beta$)]=$cx_u + \beta(ax_u - b)$ or v[P($\beta$)]=$cx_v + \beta(ax_v - b)$, **then** *Run*:= stop;
    **end**
```

It is well-known that the main difficulty with the Lagrangean approach is the selection of the sequence of multipliers, β, so that software implementations using these rules are robust. Convergence results can be found in Allen, Helgason, Kennington, and Shetty [2]. Most of the steps in the above algorithm are related to the elaborate scheme for updating β which was developed through empirical analysis.

Minor modifications can be made to the ASSIGN+1 algorithm to solve P'_1= { min cx : x ϵ T, ax=b }. Since finding a set of assignments for which ax = b may not always be possible and since from our work with the Navy Personnel Research and Development Center we have found that most constraints can be slightly violated and acceptable solutions can still be obtained, we replace ax-b = 0 with |ax-b| < ϵ. That is, instead of P'_1 we attempt to solve min { cx : x ϵ T and |ax-b| < ϵ }. For all our work we set ϵ to .01b.

IV. EMPIRICAL ANALYSIS

The algorithm ASSIGN+1 has been implemented in software and empirically analyzed on both a Sequent Symmetry S81 and an IBM 3081D for both inequality and equality side constraints. Both codes are written in Fortran and use SEMI (see Kennington and Wang [16, 17]) to solve the assignment problems. SEMI is an implementation of the shortest augmenting path algorithm for sparse semi-assignment problems and is claimed to be one of the fastest codes available for both assignment and semi-assignment problems.

We developed a test problem generator with the following inputs: (i) the number of men, (ii) the arc density, (iii) the maximum cost, $\bar{c}$, and (iv) the side constraint multiplier, k. Both the costs and the side constraint coefficients are uniformly distributed over the range $[0, \bar{c}]$. We randomly generate a feasible assignment, $\bar{x}$, and determine $\bar{b}$ for this assignment so that $a\bar{x} = \bar{b}$. The right-hand-side, b, for the side constraint is set to $k\bar{b}$. For the inequality problems and k = 1, we observed that for most problems, the side constraint was redundant and therefore a very easy problem. As k becomes smaller, the feasible region becomes smaller and for sufficiently small k the problem may become infeasible.

The generator was used to generate the eighty-one inequality problems described in Table 1. Under the column entitled "SC RHS", k was set to .2, .5, and .9 for the rows entitled "small", "med.", and "large", respectively. It should be noted that the software is very robust as a function of the magnitude of b and it requires very few iterations to satisfy the optimality criteria. Results for both normal and slope bisection are presented. Near optimal solutions to integer programs with over one-half million binary variables were routinely obtained in less than seventeen minutes.

Table 1 here

Table 2 gives our empirical results with 135 equality problems. Under the column entitled "SC RHS", k was set to .2, .5, .9, 1.2, and 1.5 for the rows entitled "v. small", "small", "med.", "large", and "v. large", respectively. The software is very robust over a wide range of input parameters and performed very well on all these problems. Though Tables 1 and 2 indicate that the quality of the solutions and the amount of time spent to obtain these solutions using normal and slope bisection are comparable we adopt the slope bisection due to the stability of the technique.

Table 2 here

In contrast with some of our previous experience using subgradient optimization, we never found a problem that caused this software any major difficulty. We attribute the robustness of this software to the elaborate scheme for updating the Lagrangean multiplier which works well for this class of problems. We attribute the speed of this software to the semi-assignment software, SEMI. Of course, the version of SEMI that we used was modified to handle single precision real cost as opposed to integer data required by the version described in Kennington and Wang [17].

Tables 3 through 6 present our empirical results comparing the specialized software for the singly constrained assignment problem with both MPSX (see Mathematical Programming System Extended [22]) and OSL (see Optimization Subroutine Library [25]). If the ASSIGN+1 software for the equality side constraint terminates due to $u-v \leq tol$ and no feasible solution has been found, then the current best known Lagrangean multiplier is used to find a solution. In this case the side constraint violation will exceed 1%. This occurred for three of sixteen problems presented in Tables 3-6. In the worst case, the maximum deviation from feasibility was 1.57%. That is, all solutions satisfied the constraint $|\mathbf{a}x-b|\leq 0.0157b$.

Tables 3, 4, 5, and 6 here

All the MPSX and OSL runs were made with default parameter settings. A few of the smaller problems were successfully solved, but the times were from two to three orders of magnitude slower than those for the specialized software. Since MPSX and OSL are general purpose integer programming systems which do not exploit the special structure of this problem we ran a specialized network with side constraint code, NETSIDE (see Kennington and Whisman [18]) on an 800x800 test problem in an attempt to solve the linear programming relaxation of this model. Convergence was not achieved after two hours of CPU time on the Sequent Symmetry S81.

We also compared the phase I specialized code of Mazzola and Neebe [23] with ASSIGN+1. These results may be found in Table 7. Under the column entitled "Side Const RHS", k was set to .2, .4, and .6 for the rows entitled "small", "medium", and "large", respectively. ASSIGN+1 was faster and produced better integer solutions on every problem attempted. Our study was restricted to dense problems because their code was designed for dense problems.

Table 7 here

We also compared ASSIGN+1 with our best alternating path code AB1 (see Kennington and Mohammadi [19]). These results can be found in Table 8. Remarkably, both codes obtained integer solutions having identical objective values for all problems attempted. ASSIGN+1 was approximately three times faster than AB1 on these test problems.

Table 8 here

As stated we have modified SEMI to handle single precision real cost coefficients as opposed to the integer cost coefficients. Generally this modification is expected to increase the execution time drastically, but in this case, as Table 9 indicates this increase was less than ten percent. Results presented in Table 9 are the average wall clock times for three pure assignment problems, running on a Sequent Symmetry S81 using the floating point accelerator.

Table 9 here

V. THE SINGLY CONSTRAINED UNBALANCED ASSIGNMENT PROBLEM

Navy personnel assignment problems are unbalanced in which the number of jobs m exceeds the number of men n. After dualizing the side constraint we obtain an unbalanced pure assignment problem whose dual is

$$\text{maximize} \quad \sum_i \lambda_i + \sum_j \pi_j \tag{10}$$

$$\hat{c}_{ij} - \lambda_i - \pi_j \geq 0, \quad (i,j) \in E \tag{11}$$

$$\pi_j \leq 0, \qquad j = 1, \ldots, m. \tag{12}$$

The dual variable, π_j, is associated with job j and the dual variable, λ_i, is associated with the man i. The dual problem for the balanced assignment problem, (1)-(4) is (10) and (11).

SEMI, the Fortran code used to solve the pure assignment problems in the previous sections of this paper is the software implementation of a shortest augmenting path algorithm developed by Kennington and Wang [17]. The algorithm is a dual method and consists of four phases: column reduction, reduction transfer, row reduction augmentation, and shortest path augmentation. In each phase both dual feasibility, $\hat{c}_{ij} - \lambda_i - \pi_j \geq 0$ for all $(i,j) \in E$ and complementary slackness $x_{ij}(\hat{c}_{ij} - \lambda_i - \pi_j) = 0$ for all $(i,j) \in E$ are maintained and the procedure works toward obtaining primal feasibility, (2) and (3). Minor modifications to SEMI were incorporated so that $\pi_j \leq 0$ and $(x_{ij} - 1)\pi_j = 0$ for all $(i,j) \in E$ were also maintained throughout the four phases. The modified code is called UNBAL_SEMI.

Table 10 presents our empirical results comparing SEMI and UNBAL_SEMI for the assignment problem and presents results for the singly constrained unbalanced assignment problem. Test runs were performed on an IBM 3081D and a Sequent Symmetry S81. Every entry in columns 2-6 of Table 10 is the average run time for three randomly generated problems except for the entries in the last row which are for a singly constrained unbalanced assignment problem provided by the Navy Personnel Research and Development Center in San Diego.

By adding dummy nodes and artificial arcs, one can always convert an unbalanced problem to a balanced one. As shown in Table 10, the specialized code for the unbalanced problem can run four times faster than the corresponding balanced code. For the 400x600 assignment problems, UNBAL_SEMI was four times faster than SEMI on problems in which 200 dummy men and 120,000 dummy arcs were appended. We also find that for this application, the IBM 3081D is approximately twice as fast as the Sequent Symmetry S81.

Table 10 here

VI. SUMMARY AND CONCLUSIONS

We have presented a new algorithm, ASSIGN+1, for the singly constrained assignment problem. This algorithm is applicable for balanced and unbalanced assignment problems having either an inequality or an equality side constraint. The algorithm uses Lagrangean relaxation and solves a series of pure sparse assignment problems.

The empirical results of test runs of the Fortran implementation of the algorithm for balanced problems with inequality and equality side constraints and unbalanced problems with an inequality side constraint indicate that these three codes are very robust and need very few iterations to satisfy the optimality criteria. Remarkably, near optimal solutions to integer programs with over one-half million binary variables are obtained in less than seventeen minutes on a Sequent Symmetry S81 using a single processor. The results from test runs of ASSIGN+1, MPSX, and OSL demonstrated the superiority of the new algorithms over state-of-the-art general purpose software and results from test runs of ASSIGN+1, AB1, and the Mazzola-Neebe code indicates that ASSIGN+1 is more than three times faster than AB1 and about forty times faster than the Mazzola-Neebe code. While integer solutions obtained by AB1 are identical to the ones obtained by ASSIGN+1, they are better than the ones obtained by phase 1 of the Mazzola-Neebe code.

REFERENCES

1. V. Aggarwal, "A Lagrangean-Relaxation Method for the Constrained Assignment Problem," *Computers and Operations Research* vol. 12 pp. 97-106, 1985.

2. E. Allen, R. Helgason, J. Kennington, and B. Shetty, "A Generalization of Polyak's Convergence Result For Subgradient Optimization," *Mathematical Programming* vol. 13 pp. 309-318, 1987.

3. M. Ball, U. Derigs, C. Hilbrand, and A. Metz, "Matching Problems with Generalized Upper Bound Side Constraints," *Networks* vol. 20 pp. 703-721, 1990.

4. R. Barr, F. Glover, and D. Klingman, "The Alternating Basis Algorithm for the Assignment Problems," *Mathematical Programming,* vol. 13 pp. 1-13, 1977.

5. R. Barr, K. Farhangian, and J. Kennington, "Networks with Side Constraints: An LU Factorization Update," *The Annals of the Society of Logistics Engineers* vol. 1 pp. 66-85, 1986.

6. J. Brans, M. Leclercq, and P. Hansen, "An Algorithm for Optimal Reloading of Pressurized Water Reactors," *Operational Research'72,* Editor M. Ross, North Holland Publishing Company: Amsterdam, pp. 417-428, 1973.

7. N. Bryson, "Parametric Programming and Lagrangian Relaxation: The Case of the Network Problem with a Single Side-Constraint," *Computers and Operations Research* vol. 18 pp. 129-140, 1991.

8. H. Everett, "Generalized Lagrange Multiplier Method for Solving Problems of Optimum Allocation of Resources," *Operations Research* vol. 11 pp. 399-417, 1963.

9. S. Gass and T. Saaty, "The Computational Algorithm for the Parametric

Objective Function," *Naval Research Logistics Quarterly* vol. 2 pp. 39–45, 1955.

10. A. Geoffrion, "An Improved Implicit Enumeration Approach for Integer Programming," *Operations Research* vol. 17 pp. 437–454, 1969.

11. A. Geoffrion, "Lagrangean Relaxation for Integer Programming," *Mathematical Programming* vol. 2 pp. 82–114, 1974.

12. F. Glover, "A Multiphase-Dual Algorithm for the Zero-One Integer Programming," *Operations Research* vol. 13 pp. 879-919, 1965.

13. F. Glover, D. Karney, D. Klingman, and R. Russell, "Solving Singly Constrained Transshipment Problems," *Transportation Science* vol. 12 pp. 277–297, 1978.

14. A. Gupta and J. Sharma, "Tree Search Method for Optimal Core Management of Pressurised Water Reactors," *Computers and Operations Research* vol. 8 pp. 263-269, 1981.

15. J. Kennington and Z. Wang, "An Empirical Analysis of the Dense Assignment Problem: Sequential and Parallel Implementations", *ORSA Journal on Computing*, vol. 3 pp. 299-306, 1991.

16. J. Kennington and Z. Wang, SEMI Users Guide, Technical Report 90–CSE–20, Department of Computer Science and Engineering, Southern Methodist University, Dallas, TX 75275, 1990.

17. J. Kennington and Z. Wang, "A Shortest Augmenting Path Algorithm for the Semi-Assignment Problem," *Operations Research*, vol. 40 pp. 178–187, 1992.

18. J. Kennington and A. Whisman, "Netside Users Guide," Technical Report 90–CSE–37, Department of Computer Science and Engineering, Southern Methodist University, Dallas, TX 75275, 1990.

19. J. Kennington and F. Mohammadi, "The Singly Constrained Assignment Problem: An AP Basis Approach," Technical Report 93-CSE-25, Department of Computer Science and Engineering, Southern Methodist University, Dallas, TX 75275, 1993.

20. D. Klingman and R. Russell, "Solving Constrained Transportation Problems," *Operations Research* vol. 23 pp. 91-106, 1975.

21. D. Klingman and R. Russell, "A Stream Lined Approach to the Singly Constrained Transportation Problem," *Naval Research Logistics Quarterly* vol. 25 pp. 681-695, 1978.

22. Mathematical Programming System Extended, Mixed Integer Programming/370 Program Reference Manual, IBM SH19-1099-1, 1979.

23. J. Mazzola and A. Neebe, "Resource Constrained Assignment Scheduling." *Operations Research* vol. 34 pp. 560-572, 1986.

24. G. Nemhauser and L. Wolesy, *Integer and Combinatorial Optimization*, John Wiley and Sons: New York, NY, 1988.

25. Optimization Subroutine Library: Guide and Reference, IBM, SC23-0519-1, 1990.

26. C. Ryu and M. Guignard, "Lagrangean Approximation Techniques for the Simple Plant Location Problem with an Aggregate Capacity Constraint." Working Paper 91-06-06, Department of Decision Sciences, The Wharton School, University of Pennsylvania, Philadelphia, PA 19104-6366, 1991.

27. R. Parker and R. Rardin, *Discrete Optimization*, Academic Press Incorporated: New York, NY, 1988.

28. H. Salkin, *Integer Programming*, Addison-wesley Publishing Company: Reading, Massachusetts, 1974.

29. J. Shapiro, "Generalized Lagrange Multipliers in Integer Programming."

Operations Research vol. 19 pp. 68–76, 1971.

30. J. Shapiro, "A Survey of Lagrangean Techniques for Discrete Optimization," *Annals of Discrete Mathematics* vol. 5 pp. 113–138, 1979.

31. G. Yu, "Algorithms for Optimizing Piecewise Linear Functions and for Degree Constrained Minimum Spanning Tree Problems," Department of Decision Sciences, The Wharton School, University of Pennsylvania, Philadelphia, PA 19104–6366, 1991.

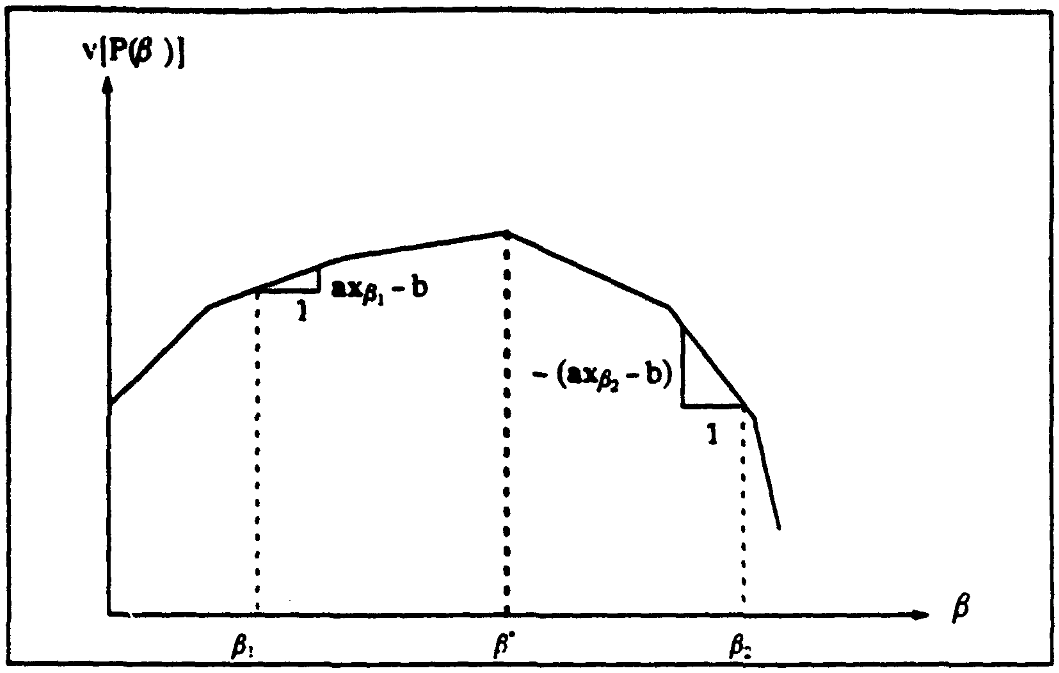


Figure 1. Unique optimum for LD_1

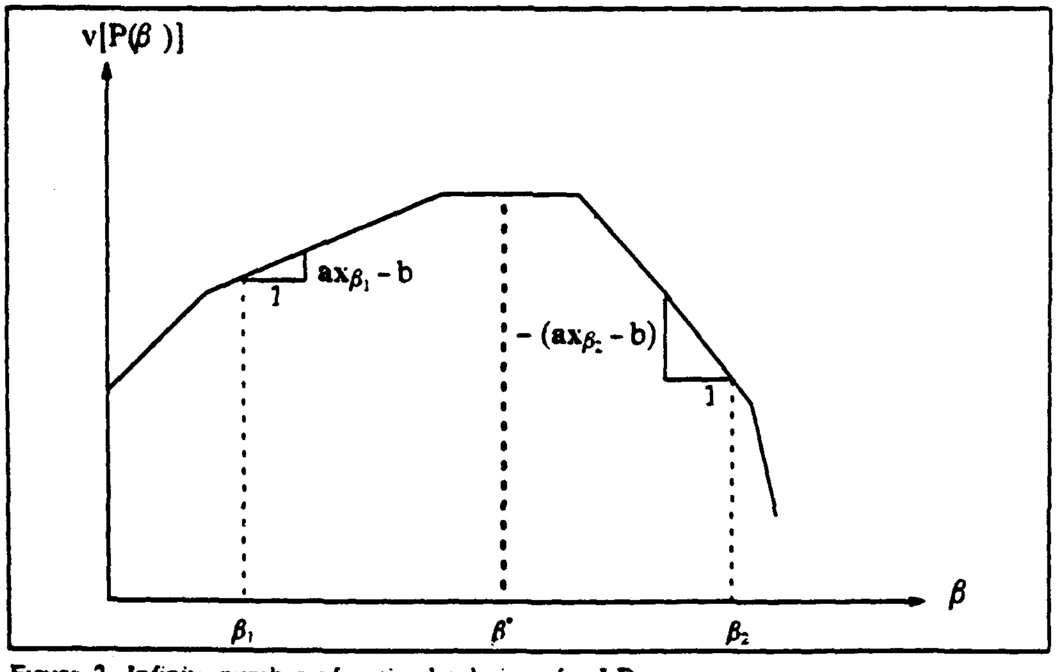


Figure 2. Infinite number of optimal solutions for LD_1

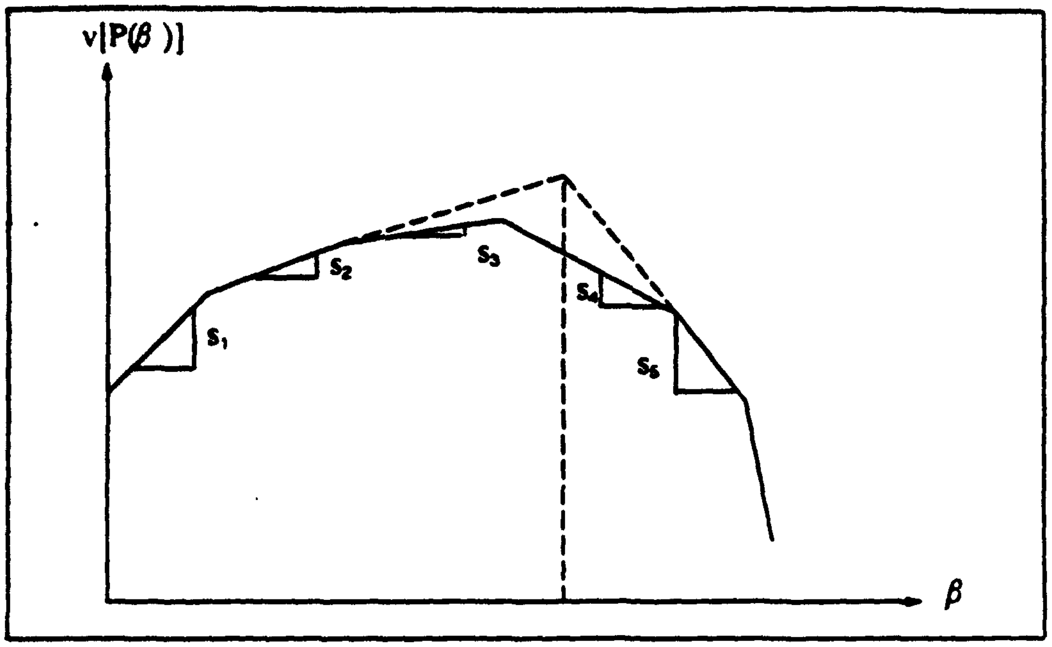


Figure 3. β selection using slope bisection

Table 1. The assignment problem with an inequality side constraint

Problem Size	C/S Range 0 to	SC RHS	Normal Bisection			Slope Bisection		
			# of Iter	Time[1] (min)	Solution % Opt.	# of Iter	Time[1] (min)	Solution % Opt.
800x800 (256,000 arcs)	1000	small	11.33	5.42	99.60	11.33	5.43	99.69
		med.	13.33	6.50	99.16	11.00	5.30	99.07
		large	8.00	3.33	99.53	7.67	3.32	99.53
	10000	small	10.33	4.98	99.48	9.33	4.40	99.48
		med.	9.67	4.30	99.85	10.33	4.85	99.48
		large	8.33	3.60	99.78	7.67	3.32	99.43
	100000	small	10.33	4.79	99.52	9.33	4.44	99.52
		med.	8.67	3.93	99.49	8.67	3.97	99.50
		large	8.67	3.71	99.62	7.67	3.26	99.57
1000x1000 (400,000 arcs)	1000	small	11.00	8.19	99.32	10.67	8.04	99.42
		med.	11.33	8.55	99.42	11.00	8.55	99.43
		large	7.00	5.42	99.39	6.67	5.24	99.39
	10000	small	10.33	8.18	99.69	10.67	8.43	99.79
		med.	10.33	8.01	99.43	9.00	7.09	99.64
		large	7.00	5.62	99.16	6.33	5.22	99.28
	100000	small	10.33	7.94	99.73	11.00	8.70	99.73
		med.	9.00	6.78	99.65	9.00	6.87	99.65
		large	7.00	5.64	99.25	6.33	5.20	99.25
1200x1200 (576,000 arcs)	1000	small	11.00	12.40	99.31	10.00	11.44	99.32
		med.	12.33	14.28	99.48	11.00	12.75	99.65
		large	8.00	9.67	99.81	11.67	14.49	99.92
	10000	small	11.67	14.07	99.40	11.00	13.43	99.26
		med.	12.67	16.04	99.64	11.00	13.10	99.65
		large	9.33	11.36	99.47	12.00	15.99	99.95
	100000	small	12.33	14.94	99.44	11.00	13.56	99.30
		med.	13.00	15.65	99.48	11.00	12.91	99.66
		large	7.33	8.71	99.23	11.00	13.71	99.83
Average			9.63	8.01	99.49	9.44	8.07	99.56

[1] Times are wall clock time on a Sequent Symmetry S81 using one processor.

Table 2. The assignment problem with an equality side constraint

Problem Size	C/S Range 0 to	SC RHS	Normal Bisection			Slope Bisection		
			iter#	Time[1] (min)	%opt	iter#	Time[1] (min)	%opt
	1000	v. small	10.00	4.74	99.35	9.33	4.46	99.56
		small	11.67	5.88	100.00	9.67	4.57	99.96
		med.	8.00	3.78	99.90	7.67	3.58	99.84
		large	9.33	4.35	99.70	9.67	4.61	99.70
		v. large	9.67	4.55	99.33	8.67	4.02	98.73
800x800 (256,000 arcs)	10000	v. small	10.67	5.13	99.70	11.33	5.60	99.87
		small	10.33	5.15	100.00	9.33	4.45	99.91
		med.	7.00	3.40	99.93	7.00	3.39	99.81
		large	9.67	4.66	99.69	10.00	4.79	99.74
		v. large	9.67	4.67	99.75	9.67	4.61	99.79
	100000	v. small	11.33	5.54	99.83	11.33	5.49	99.94
		small	11.00	5.48	100.00	8.67	4.06	99.74
		med.	7.67	3.81	99.92	8.00	4.04	99.85
		large	11.00	5.27	99.52	10.33	4.91	99.73
		v. large	9.67	4.70	99.76	9.67	4.59	99.84
	1000	v. small	8.33	6.69	99.70	9.00	7.45	99.49
		small	11.00	9.03	99.61	9.67	7.93	99.61
		med.	8.33	6.64	99.99	8.33	6.82	99.90
		large	11.33	9.13	99.79	10.33	8.36	99.97
		v. large	10.00	8.12	98.61	9.33	7.45	97.90
1000x1000 (400,000 arcs)	10000	v. small	8.33	6.84	99.57	10.33	8.69	99.74
		small	9.67	7.87	100.00	10.33	8.49	99.79
		med.	8.67	7.12	99.93	8.67	7.22	99.98
		large	11.67	9.96	99.74	10.00	8.11	99.31
		v. large	10.00	8.16	98.34	8.67	6.93	98.57
	100000	v. small	8.33	7.09	99.59	9.67	8.16	99.70
		small	10.00	8.09	100.00	9.33	7.77	100.00
		med.	8.67	7.22	99.96	8.67	7.15	99.96
		large	11.67	9.96	99.74	10.00	8.18	99.46
		v. large	10.00	8.08	98.34	8.67	6.92	98.65
	1000	v. small	9.67	12.61	99.84	10.33	13.18	100.00
		small	9.67	11.94	99.41	9.00	10.89	99.56
		med.	8.67	10.79	99.93	8.33	10.24	99.99
		large	11.67	14.25	99.83	10.00	11.52	99.98
		v. large	10.00	12.19	98.87	8.67	10.18	98.62
1200x1200 (576,000 arcs)	10000	v. small	9.00	11.54	99.71	10.00	13.00	99.71
		small	9.33	11.40	99.38	10.00	12.14	99.84
		med.	8.00	10.27	99.91	8.00	10.41	99.96
		large	11.67	14.52	99.65	11.00	13.68	99.87
		v. large	10.00	12.47	99.02	8.00	9.69	99.95
	100000	v. small	9.00	11.93	99.72	10.00	12.79	99.72
		small	11.33	14.18	99.58	9.00	10.94	99.84
		med.	8.00	10.35	99.93	8.33	11.00	99.96
		large	11.67	14.76	99.78	10.00	12.40	99.83
		v. large	10.00	12.56	99.06	8.00	9.76	99.72
Average			9.96	8.59	99.65	9.31	7.88	99.73

[1]Times are wall clock time on a Sequent Symmetry S81 using one processor.

Table 3. Comparisons of specialized algorithms with MPSX and OSL on an IBM 3081D (very small value of b in side constraint)

Problem Name	A100+1		A200+1		A300+1		A400+1	
Problem Characteristics								
Rows	201	201	401	401	601	601	801	801
Columns	4,000	4,000	16,000	16,000	36,000	36,000	64,000	64,000
Side Constraint Type	≤	=	≤	=	≤	=	≤	=
MPSX								
Nodes in BAB Tree	602	1,014	125	77	Too	Too	Too	Too
Iter	13,176	12,248	19,325	14,336	many	many	many	many
Time (Sec)	815	901	3,343	1,864	binary	binary	binary	binary
Obj. Value Best Inc.	31,578	33,285	NA	NA	variables	variables	variables	variables
Status	Optimal	Feasible[1]	Unknown[1]	Unknown[2]	Unknown[3]	Unknown[3]	Unknown[3]	Unknown[3]
OSL								
Nodes in BAB Tree	1,465	2,126	200	200	200	200	Error	Error
Iter	10,916	36,902	4,827	3,969	10,082	6,405	during	during
Time (Sec) ≥	1,958	564	2,038	943	3,634	3,909	input	input
Obj. Value Best Inc.	31,564	NA	NA	NA	NA	NA	NA	NA
Status	Optimal	Unknown[4]	Unknown[5]	Unknown[5]	Unkown[5]	Unknown[5]	Unknown[6]	Unknown[6]
ASSIGN+1								
Time (Sec)	0.96	0.94	6.96	5.64	20.87	13.38	45.81	37.66
Obj. Value	34,359	30,830	33,361	32,024	30,550	30,550	32,026	31,439
Dev. from Feasibility	0%	1.57%	0%	0.45%	0%	0.98%	0%	0.28%
Max. Dev. from Opt.	9.35%	0%	0.45%	0.46%	0.29%	1.06%	1.58%	0%

[1] Terminated due to node table overflow after obtaining an incumbent but prior to obtaining an optimal solution.
[2] Manually stopped after 30 minutes of CPU time prior to obtaining the first incumbent.
[3] MPSX/370 restricts the number of integer variables to maximum of 32,767.
[4] Terminated due to basis file overflow.
[5] Terminated due to branch and bound node table overflow (node table set to 200 problems).
[6] Input error due to file size (file has 193,481 lines).

Table 4. Comparisons of specialized algorithms with MPSX and OSL on an IBM 3081D (small value of b in side constraint)

Problem Name	A100+1		A200+1		A300+1		A400+1	
Problem Characteristics								
Rows	201	201	401	401	601	601	801	801
Columns	4,000	4,000	16,000	16,000	36,000	36,000	64,000	64,000
Side Constraint Type	≤	=	≤	=	≤	=	≤	=
MPSX								
Nodes in BAB Tree	1,009	1,009	728	915	Too many binary variables	Too many binary variables	Too many binary variables	Too many binary variables
Iter	6,332	4,082	16,324	19,153				
Time (Sec)	320	308	1,789	2,212				
Obj. Value Best Inc.	NA	NA	NA	NA				
Status	Unknown[2]	Unknown[1]	Unknown[2]	Unknown[2]	Unknown[3]	Unknown[3]	Unknown[3]	Unknown[3]
OSL								
Nodes in BAB Tree	715	2,054	200	200	200	200	Error during input	Error during input
Iter	5,364	17,261	2,087	2,152	3,097	3,041		
Time (Sec) ≥	907	223	1,187	975	3,338	3,350		
Obj. Value Best Inc.	6,291	NA	NA	NA	NA	NA	NA	NA
Status	Optimal	Unknown[4]	Unknown[5]	Unknown[5]	Unkown[5]	Unknown[5]	Unknown[6]	Unknown[6]
ASSIGN+1								
Time (Sec)	1.44	1.14	6.38	4.87	19.73	16.69	42.55	33.05
Obj. Value	6,376	6,624	5,564	5,564	6,022	5,923	6,200	6,200
Dev. from Feasibility	0%	0.03%	0%	0.93%	0%	0.74%	0%	0.81%
Max. Dev. from Opt.	1.76%	0%	0.87%	0.94%	0.97%	0%	0.82%	0%

[1] Terminated due to node table overflow prior to obtaining the first incumbent.
[2] Terminated due to problem file overflow prior to obtaining the first incumbent .
[3] MPSX/370 restricts the number of integer variables to maximum of 32,767.
[4] Terminated due to basis file overflow.
[5] Terminated due to branch and bound node table overflow (node table set to 200 problems).
[6] Input error due to file size (file has 193,481 lines).

Table 5. Comparisons of specialized algorithms with MPSX and OSL on an IBM 3081D (large value of b in side constraint)

Problem Name	A100+1		A200+1		A300+1		A400+1	
Problem Characteristics								
Rows	201	201	401	401	601	601	801	801
Columns	4,000	4,000	16,000	16,000	36,000	36,000	64,000	64,000
Side Constraint Type	≤	=	≤	=	≤	=	≤	=
MPSX								
Nodes in BAB Tree	1,018	1,009	43	915	Too many binary variables	Too many binary variables	Too many binary variables	Too many binary variables
Iter	4,129	7,847	13,911	17,135				
Time (Sec)	428	470	499	1,538				
Obj. Value Best Inc.	3,992	NA	3,763	NA				
Status	Feasible[1]	*Unknown*[2]	*Optimal*	*Unknown*[2]	Unknown[3]	Unknown[3]	Unknown[3]	Unknown[3]
OSL								
Nodes in BAB Tree	10	2,584	94	200	200	200	Error during input	Error during input
Iter	NA	32,992	2,109	3,104	5,798	9,905		
Time (Sec) ≥	NA	4,637	908	1,580	5,440	4,885		
Obj. Value Best Inc.	3,911	NA	3,763	NA	3,744	NA	NA	NA
Status	Feasible[4]	Unknown[5]	Unknown[5]	Unknown[5]	Unkown[5]	Unknown[5]	Unknown[6]	Unknown[6]
ASSIGN+1								
Time (Sec)	1.48	1.35	7.41	4.52	18.99	17.55	39.45	27.24
Obj. Value	3,964	3,863	3,765	3,757	3,743	3,723	3,952	3,952
Dev. from Feasibility	0%	1.05%	0%	0.74%	0%	1.01%	0%	0.45%
Max. Dev. from Opt.	1.05%	0%	0.08%	0.28%	0.30%	0%	0.11%	0.11%

[1] Terminated due to node table overflow after obtaining the first incumbent but prior to obtaining an optimum.
[2] Terminated due to problem file overflow prior to obtaining the first incumbent.
[3] MPSX/370 restricts the number of integer variables to maximum of 32,767.
[4] Terminated due to OSL data overflow.
[5] Terminated due to basis file overflow.
[6] Terminated due to branch and bound node table overflow (node table set to 200 problems).
[7] Input error due to file size (file has 193,481 lines).

Table 6. Comparisons of specialized algorithms with MPSX and OSL on an IBM 3081D (very large value of b in side constraint)

Problem Name	A100+1		A200+1		A300+1		A400+1	
Problem Characteristics								
Rows	201	201	401	401	601	601	801	801
Columns	4,000	4,000	16,000	16,000	36,000	36,000	64,000	64,000
Side Constraint Type	$\leq$	=	$\leq$	=	$\leq$	=	$\leq$	=
MPSX								
Nodes in BAB Tree	1	1,204	1	138	Too many binary variables	Too many binary variables	Too many binary variables	Too many binary variables
Iter	3,217	16,283	7,665	22,048				
Time (Sec)	37	1,103*	199	1,641				
Obj. Value Best Inc.	3,852	7,030	3,691	NA				
Status	Optimal	Feasible[1]	Optimal	Unknown[2]	Unknown[3]	Unknown[3]	Unknown[3]	Unknown[3]
OSL								
Nodes in BAB Tree	0	2,318	1	5	0	6	Error during input	Error during input
Iter	253	24,228	481	NA	794	NA		
Time (Sec) $\geq$	16	3,530	100	NA	365	NA		
Obj. Value Best Inc.	3,852	NA	3,691	NA	3,696	NA	NA	NA
Status	Optimal	Unknown[4]	Optimal	Unknown[5]	Optimal	Unknown[5]	Unknown[6]	Unknown[6]
ASSIGN+1								
Time (Sec)	0.12	1.18	0.55	5.31	1.20	15.26	3.30	24.50
Obj. Value	3,852	7,023	3,691	6,503	3,969	7,437	3,896	6,714
Dev. from Feasibility	0%	0.91%	0%	0.60%	0%	0.45%	0%	0.65%
Max. Dev. from Opt.	0%	3.27%	0%	0%	0%	1.77%	0%	0%

[1] Terminated due to node table overflow after obtaining the first incumbent but prior to obtaining an optimum.
[2] Manually stopped after 10 hours of wall clock time and 25 minutes of CPU prior to obtaining the first incumbent.
[3] MPSX/370 restricts the number of integer variables to maximum of 32,767.
[4] Terminated due to basis file overflow.
[5] Terminated due to OSL data overflow.
[6] Input error due to file size (file has 193,481 lines).

Table 7. The empirical comparison of the Mazzola-Neebe phase I code and ASSIGN+1 solving NxN 100% dense singly constrained assignment problems.

N	Arcs	Side Const RHS	Mazzola-Neebe		ASSIGN+1		
			Obj Value	Time[1] (Sec)	Obj Value	Time[1] (Sec)	Solution % Opt.
40	1,600	small	10,446.1	10.33	6,351.6	0.61	94.43
		medium	4,414.4	12.73	2,991.2	0.56	95.85
		large	2,285.1	11.37	2,143.8	0.61	94.41
60	3,600	small	12,225.2	50.45	6,571.7	1.67	95.72
		medium	5,667.1	54.30	3,225.8	1.83	97.53
		large	2,859.4	46.50	2,248.3	2.00	97.57
80	6,400	small	13,867.4	120.92	6,638.4	3.44	96.42
		medium	5,578.7	148.81	3,264.3	3.29	96.28
		large	2,616.1	152.02	2,301.8	3.96	97.46
100	10,000	small	12,759.3	331.37	6,271.7	6.09	96.54
		medium	5,331.8	303.62	3,135.1	7.18	98.25
		large	2,529.0	313.83	2,222.2	6.76	97.23

[1] Times are wall clock time on a Sequent Symmetry S81 using one processor.

Table 8. The empirical comparison of AB1 and ASSIGN+1 solving NxN 30% dense singly constrained assignment problems.

N	Arcs	Side Const RHS	AB1			ASSIGN+1		
			Time[1] (Sec)	Obj Value	Solution % Opt.	Time[1] (Sec)	Obj Value	Solution % Opt.
200	12,000	small	68.76	43,326.9	99.04	18.66	43,326.9	98.31
		medium	59.11	22,441.8	98.79	19.12	22,441.8	98.62
		large	50.16	23,223.1	99.39	18.80	23,223.1	99.22
300	27,000	small	201.22	42,604.7	98.75	50.57	42,604.7	98.57
		medium	177.90	29,535.0	99.13	51.62	29,535.0	98.98
		large	138.15	22,880.2	99.91	47.98	22,880.2	99.44
400	48,000	small	430.2	42,490.9	98.67	105.69	42,490.9	98.50
		medium	378.5	28,797.1	99.29	98.90	28,797.1	99.02
		large	326.2	23,208.0	99.66	94.27	23,208.0	99.46
500	75,000	small	757.3	44,025.2	99.32	177.53	44,025.2	99.25
		medium	686.9	29,480.1	99.48	158.59	29,480.1	99.35
		large	566.0	22,746.8	99.67	167.50	22,746.8	99.63

[1] Times are wall clock time on a Sequent Symmetry S81 using one processor.

Table 9. Comparison of the integer and real versions of pure assignment codes. (The Weitek floating point accelerator was activated in all runs.)

Problem Size	400x400	800x800	1000x1000	1200x1200
SEMI (Secs.) (integer)	3.66	14.93	26.36	37.26
ASSIGN+1 (Secs.) (floating point)	4.00	16.38	28.37	40.71
Increase for floating point arithmetic	9.56%	9.71%	7.62%	9.25%

Table 10. CPU times (sec.) on an IBM 3081D and wall clock times (sec.) on a Sequent Symmetry S81 for balanced and unbalanced pure assignment problems and singly constrained unbalanced assignment problems.

Size	**Assignment**				**Assignment+1 Side Constraint**	
	SEMI[1]		UNBAL_SEMI		UNBAL_ASSIGN+1	
	IBM	Sequent	IBM	Sequent	IBM	Sequent
100x100	0.11	0.19	0.14	0.28	1.89	3.31
100x200	0.44	1.28	0.07	0.16	1.39	2.59
100x300	1.79	3.70	0.14	0.21	2.21	4.17
200x200	0.44	0.69	0.63	1.18	6.84	9.51
200x300	1.07	2.16	0.22	0.49	4.76	8.83
200x400	2.36	5.13	0.26	0.59	5.86	11.39
300x300	1.22	2.14	1.74	3.37	15.76	32.78
300x400	1.70	3.35	0.52	1.15	9.92	18.90
300x500	3.22	6.91	0.53	1.18	10.84	25.46
400x400	2.30	4.18	3.78	7.86	37.83	74.09
400x500	2.23	4.63	0.88	1.92	16.02	31.86
400x600	3.68	8.80	0.90	2.00	16.90	35.72
98x362[2]	NA	NA	NA	NA	0.28	0.59
Total	20.56	43.16	10.04	20.39	130.50	258.74

[1] dummy men nodes and artificial arcs are added to balance men and jobs.
[2] real problem provided by the Navy.

Technical Report 93-CSE-34

Solution of Convex Cost Network Flow Problems Via Linear Approximation

by

Richard V. Helgason
Rajluxmi V. Murthy

Department of Computer Science and Engineering
Southern Methodist University
Dallas, Texas 75275-0122

May 1993

This work was supported in part by the Air Force Office of Scientific Research under Grant Number AFOSR F49620-93-1-0091 and the Office of Naval Research under Grant Number N00014-92-1619.

Solution of Convex Cost Network Flow Problems Via Linear Approximation.

ABSTRACT. This work presents a new approximation approach to solving minimum cost network flow problems with nonlinear cost functions. Approximation procedures are typically based on *local* approximations, the reason being that global approximations have to face a trade off between accuracy and the resulting problem size. We propose a *global* approximation procedure which leads to the optimal solution through refinements in the region local to the solution at a given iteration. The resulting problem at each iteration is solved efficiently, without reformulation. Thus the issue of increased problem size that is associated with global approximations is resolved. Computational results demonstrate that this method takes only 2%-30% of the time taken by a traditional method for solving such problems, such as the Franke-Wolfe. The problems tested were generated randomly.

1. Introduction

Consider the following network flow problem:

$$\begin{aligned} \min\quad & f(x) \\ \text{s.t.}\quad & Ax = b \qquad\qquad (P) \\ & 0 \le x \le u \end{aligned}$$

where $f(x)$ is a continuous and convex function defined over a closed convex feasible region $F \subset \Re^n$. F lies in the positive orthant and is determined by the vector of bounds $u \in \Re^n$, the node-arc incidence matrix $A \in \Re^{m \times n}$, and the requirements vector $b \in \Re^m$. If $f(x) = \sum_{i=1}^{n} f_i(x_i)$ where $f_i(x_i)$ is a convex function for all i, we have a separable convex cost network flow problem.

Problems with the structure defined by (P) arise in several engineering and economic applications. In general, when increased flow in an arc results in an increased burden or congestion, the cost on the arc can be approximated by a convex function of the flow on that arc. Some examples of such problems are quadratic data fitting [1], water supply applications [4], electrical networks [7], equilibrium export-import trade models [12], production scheduling with overtime costs [8], and traffic congestion models [3]. Other applications (see also [17] and [25]) arise in personnel assignment and logistics [30] and stochastic programming [9] and [33]. It is also possible to convert certain problems in nonseparable form to those which have a separable form [14] and [32].

The numerous applications and the specialized structure have generated considerable interest in the problem (P). The problem can be solved by any algorithm for general nonlinear programming problems. Several authors have applied such general purpose methods to (P). Examples of such applications are the Frank-Wolfe method by Collins et al. [4] and LeBlanc et al. [23], the convex simplex method by Collins et al. [4] and Helgason and Kennington [16], the Newton method by Klincewicz [19], [20] and Dembo [10], the reduced gradient algorithm by Dembo and Klincewicz [11] and Beck et al. [2], simplicial decomposition by Hearn et al. [15], and relaxation methods by Bertsekas et al. [5], [6] and Zenios and Mulvey [34]. Problems with a quadratic objective function have been solved by the conjugate gradient method [22], Lagrangian dual coordinatewise maximization [28], and piecewise linear approximation [26].

We are interested in a piecewise linear approach for solving the problem (P). Such an approximation will yield a linear problem which can be solved efficiently and allow exploitation of the underlying structure to the fullest. Thakur [31] and Geoffrion [13] have developed bounds on the objective function value of (P) for such approximations. Collins et al. [4] have used this approach to solve water supply applications of (P). Meyer [24], [25] has discussed and implemented a two-segment approximation. His approach is to approximate the objective function *locally*. The reasoning being that *global* approximations have the curse of resulting in the need to solve problems with increased dimensions, and therefore, force a tradeoff between the accuracy of the approximation and the size of the resulting problem to be solved.

Our approach is to approximate the objective function *globally* and refine it *locally* in successive iterations. but we solve the resulting piecewise linear problem with a *direct* method. For a discussion of the method and its implementation see [27]. The application of this method resolves the dilemma of increased dimensions since it does not require a reformulation of the resulting approximation, and therefore, the dimension of the problem to be solved is no more than the dimension of the original problem (P). The fact that reformulation is not required also enables us to avoid expensive data manipulation after each successive approximation. We discuss in detail the approximation and the solution procedure for the problem (P) in the next section. outline the salient points of the implementation in section 3. present computational results in section 4 and close the presentation with a summary of the results.

2. The Solution Methodology

2.1 *The Approximation Procedure.*

The global approximation to

$$f(x) = \sum_{j=1}^{n} f_j(x_j)$$

is given by:

$$g(x) = \sum_{j=1}^{n} g_j(x_j)$$

where

$$g_j(x_j) = \begin{cases} c_j^1, & \text{if } 0 \le x_j \le u_j^1; \\ \vdots \\ c_j^{s_j}, & \text{if } u_j^{s_j-1} \le x_j \le u_j^{s_j}, \end{cases}$$

and s_j is the number of segments or grid points that arc e_j is allowed to have. $u_j^1, \ldots, u_j^{s_j}$ are the *breakpoints* such that $0 < u_j^1 < u_j^2 \ldots < u_j^{s_j} = u_j$ (the capacity of the arc e_j). The respective costs $c_j^1, \ldots, c_j^{s_j}$ of the s_j segments are the slopes of the line segments joining the endpoints. and since $f(x)$ is convex they form an ascending sequence. Therefore.

$$0 < c_j^1 < c_j^2 \ldots < c_j^{s_j} < \infty$$

The approximation results in the following piecewise linear problem:

$$\begin{aligned} \min \quad & \sum_{j,k} c_j^k x_j^k \\ \text{s.t.} \quad & \sum_{j,k} a_{ij} x_j^k = b_i \\ & 0 \leq x_j^k \leq u_j^k \end{aligned} \qquad (\bar{P})$$

where $i = 1, \ldots, m$, $j = 1, \ldots, n$ and $k = 1, \ldots, s_j$.

The approximation requires only function evaluations and no information about the derivative of the cost function.

Note that since $f(x)$ is convex it is dominated by $g(x)$ on the entire domain, i.e.

$$f(x) \leq g(x), \quad \forall x \in F$$

Clearly, an optimal solution x^* to $(\bar{P})$ is a feasible solution for (P) and $f(x^*)$ gives an upper bound on the optimal objective function value $v(P)$. Therefore, if x^{**} is an optimal solution for (P), then $f(x^{**}) \leq f(x^*)$. The problem $(\bar{P})$ is solved using RESBAS [27], which is a FORTRAN implementation of the aforementioned direct approach to solving piecewise linear problems. The objective function of (P) evaluated at x^* is compared with a lower bound which provides a termination criterion discussed in detail in section (2.3). The comparison determines whether an ϵ-optimal solution has been obtained, where ϵ is a prespecified tolerance parameter for the objective function value. Thus, at termination the objective function value is guaranteed to be within ϵ percent of the optimal value for the problem (P).

2.2 *Refinement of the Approximation*

If the optimal solution to $(\bar{P})$ does not yield the desired ϵ-optimal solution to (P), the approximation is refined as follows. Since the flows in the arcs are expected to be *approaching* the optimal solution, *breakpoints* not in the vicinity of the current solution are judged to be of little value. Therefore, the approximation is refined in the neighbourhood of the current solution point, maintaining a constant number of grid points. As opposed to a fixed grid approach [4] the grid size is reduced with each refinement, and successive approximations are increasingly accurate, *locally*, for each arc.

To further illustrate the refinement process, let each arc have s_j *breakpoints* without loss of generality. If the flow on arc e_j is found to be at the k^{th} segment, the distance d to be refined is determined as follows:

$$d = \begin{cases} [u_j^{k-2}, u_j^{k+1}], & \text{if } 2 < k < s_j - 1; \\ [0, u_j^3], & \text{if } k \leq 2; \\ [u_j^{s_j-2}, u_j^{s_j}] & \text{if } k \geq s_j - 1. \end{cases}$$

This interval is divided into ($s_j - 2$) equal segments, and the remaining two *breakpoints* are set at

$$\begin{cases} u_j^{k-2}, u_j^{s_j}, & \text{if } 2 < k < s_j - 1; \\ u_j^3 + 0.5(u_j^{s_j} - u_j^3), u_j^{s_j}, & \text{if } k \leq 2; \\ 0.5u_j^{s_j-2}, u_j^{s_j-2} & \text{if } k \geq s_j - 1. \end{cases}$$

Thus, each successive approximation results in a refined grid in the vicinity of the current solution, and yields two *long* segments in the portion where the solution does not currently lie. In case the solution to the refined approximation is found to lie in the currently unrefined portion of the arc, the strategy allows the refinement to *slide over* to the relevant portion. This process, therefore, enhances accuracy in the neighbourhood where the solution is expected to lie, along with providing the flexibility of redefining the region to be refined if the need arises. It also addresses the concern that shrinking the intervals too rapidly may lead to the final solution being unable to meet the desired accuracy [17].

2.3 *The Stopping Criterion*

At each iteration $f(x^*)$ yields an upper bound to $f(x^{**})$. In addition we need a lower bound to determine whether the desired accuracy has been achieved. We developed lower bounds using two methods, namely, a Frank-Wolfe approach and a Lagrangian approach and carried out a comparative study of their computational performance.

Frank-Wolfe Bound

Proposition 2.1 *Let y^* be an optimal solution to the problem*

$$\begin{aligned} \min \quad & \nabla f(x^*)y \\ \text{s.t.} \quad & Ay = b \\ & 0 \leq y \leq u \end{aligned} \tag{2.1}$$

where x^* is an optimal solution for the problem $(\tilde{P})$ at a given iteration. Then

$$f(x^{**}) \geq f(x^*) + \nabla f(x^*)(y^* - x^*) \tag{2.2}$$

Proof: See [18]

This approach requires the evaluation of the first derivative of $f(x)$. It is worth noting that if $f(x)$ is a quadratic function, the search for an optimal step size required in the solution of (2.1) is greatly simplified.

Lagrangian Bound

Consider the Lagrangian relaxation of (P)

$$\min \quad f(x) + \pi(Ax - b) \tag{2.3}$$

$$0 \leq x \leq u$$

When $f(x)$ is a separable function as in this case a bound based on the Lagrangian relaxation is ideal since the problem (2.3) decomposes into n single variable subproblems. Let $\bar{x}$ be the optimal solution to (2.3). Then since (2.3) is a relaxation of (P)

$$f(x^{**}) \geq f(\bar{x}) \tag{2.4}$$

For our purpose we use the duals obtained by solving $(\tilde{P})$ for the vector π in (2.3).

Proposition 2.2 In the special case where $f(x)$ is a separable quadratic function, the solution procedure for (2.3) is greatly simplified and the optimal objective function value is given by:

$$\sum_{i=1}^{n} \frac{-\bar{x}_i^2}{4q_i} + \pi b, \quad \text{if } 0 \leq x_i \leq u_i;$$

$$q_i u_i^2 - (\pi_{F(i)} - \pi_{T(i)} - l_i)u_i + \pi b, \quad \text{if } x_i > u_i \tag{2.5}$$

where $x_i = (\pi A_{.i} - l_i)/2q_i$. $A_{.i}$ is the i^{th} column of A. For arc e_i, l_i and q_i are the linear and quadratic cost coefficients respectively. $F(i), T(i)$ are its *from* and *to* nodes, and

$$\bar{x}_i = \begin{cases} x_i, & \text{if } 0 \leq x_i \leq u_i; \\ 0, & \text{if } x_i < 0; \\ u_i & \text{if } x_i > u_i \end{cases}$$

Proof: Since $f_i(x_i) = l_i x_i + q_i x_i^2$, therefore, $x_i = (\pi A_{.i} - l_i)/2q_i$ gives a stationary point, which will also be the local and global minimum point, for (2.3). Further, since $A_{.i}$ represents the i^{th} arc of the network, therefore, $x_i = (\pi_{F(i)} - \pi_{T(i)} - l_i)/2q_i$, and can be readily evaluated. This value of x_i is, however, not guaranteed to satisfy $0 \leq x_i \leq u_i$. In case it does not lie within the prescribed bounds, since $f(x)$ is convex, the minimum will occur at the bound closest to this infeasible value of x_i. Hence, $\bar{x}_i$ as defined above is the value at which the minimum of (2.3) occurs and the corresponding optimal objective function value is given by (2.5).

Even if $f(x)$ is not a quadratic function, the objective function of (2.3) though nonlinear, is separable. Therefore, solving (2.3) is equivalent to solving n single-variable nonlinear-cost problems with the only constraints being the bound on the variable. Such problems are not difficult to solve.

2.4 *The Algorithm*

STEP 0

Approximate (P) by $\tilde{P}_0$. Divide arc e_j into s_j segments. Calculate the cost on each segment, given by the slope of the line segment joining its end points.

Set *lowerbound* $\leftarrow$ -BIG, where BIG is a large number.

Define ϵ, the error tolerance and set $k = 0$.

STEP 1

Use RESBAS [27] to solve $\tilde{P}_k$. Let x_k^* be the solution.

STEP 2

Evaluate $f(x_k^*)$ and obtain a lower bound *lbd* by the Frank-Wolfe or the Lagrangian method.

If *lbd* > *lowerbound*, then *lowerbound* → *lbd*.

If

$$\frac{f(x_k^*) - lowerbound}{|lowerbound|} \leq \text{tol}$$

then

stop with x_k^* as the ϵ-optimal solution and $f(x_k^*)$ as the objective function value

else

continue.

STEP 3

$k \leftarrow k+1$. Approximate the problem (P) by $\tilde{P}_k$.

Go to step 1.

3. The Implementation

RESBAS has been modified and extended to implement the algorithm. The *integer* FORTRAN implementation is called PWFW or PWLAG, depending on whether the Frank-Wolfe or the Lagrangian approach is used for obtaining the lower bound.

The lower bound is obtained using one of the two approaches in the subroutine BOUND. Simultaneously, the cost function $f(x)$ for P is evaluated at x_k^* and this gives the upper bound at the iteration k. The refinement of the approximation, discussed in (2.2) is carried out in the subroutine SHIFT. The data structures are an extension of those used for RESBAS [27]. Two additional arc length arrays store the linear and quadratic cost coefficients for each arc. SHIFT has an additional node length array *curcst* which facilitates the updating of the duals after each refinement. This is necessary since the effective cost of the basic arcs is likely to change with a change in the approximating function and these changes are needed irrespective of which lower bound technique is used.

The subroutine BOUND requires only one additional node length array for PWLAG which is called REQ. This array is used to store the requirements vector b, which is needed for calculating the lower bound. Many more arrays are needed for BOUND in PWFW. In order to solve (2.1) an implementation of a pure network problem solver is needed. We use NETFLO [19] here, which is an efficient implementation of the upper-bounded simplex method on the graph. NETFLO uses six basic node-length arrays and we have added these to RESBAS. In addition, an arc-length array stores the capacity vector u, and two arc-length arrays, GRADCST and TOTFLW, are used to store the gradient vector $\nabla f(x_k^*)$ and the direction $(y^* - x_k^*)$, respectively.

The arcs are stored in a backward star format. The starting solution for (2.1) in the first iteration is the same as that for solving $(\tilde{P})$, which is the advanced start solution in NETFLO. Subsequently, the optimal solution obtained for (2.1) in a given iteration is used as the starting solution in the next one.

The implementation utilizes an eight segment approximation on each arc. With the results obtained in [27], this was thought to be a suitable number of segments. However, the implementation can be readily extended to use a greater number of segments.

4. Computational Results

Problem Generation

Due to the lack of a standard problem generator for generating problems of the form (P), we developed one based on the method of Ravindran and Lee [29]. For a given solution point x, a corresponding dual vector π, a desired constraint matrix A and a quadratic cost matrix Q, a right hand side vector b and a vector of linear cost coefficients l are generated such that the point x is a Kuhn-Tucker point. If Q is positive semi-definite, x is guaranteed to be a globally minimum point for the problem. Though the problems thus generated have quadratic costs, the solution methodology is capable of handling general convex cost problems.

For our purpose, NETGEN [21] is used to generate a problem of specified dimensions. The problem is then solved using NETFLO [18], and the optimal solution thus obtained is used as x, the point at which the global minimum for the quadratic cost problem will occur. Q is chosen to be a diagonal matrix with positive elements. This ensures that a separable convex cost network flow problem, with a known optimal solution and minimum value, is generated. In the case where the resulting problem has very large linear coefficients the problem is rejected since the linear coefficients will dominate the optimal solution to this problem while we are interested in quadratic cost problems.

For the purpose of comparison we implemented the Frank-Wolfe (FW) algorithm, a classical method for solving general convex cost problems. Since the subproblems to be solved for FW are linear network flow problems, a modification of NETFLO is used to solve these.

Problem Description

The problems that were generated have number of nodes varying from 50-600 and the number of arcs varying from 200-15,000. The first set of problems tested have a cost range varying from 1-50. The percentage of arcs with high cost is fixed at 30, and the number of capacitated arcs is either 20, 25 or 80. The capacity range varies between 100-500 to 10,000-15,000. This is a typical parameter specification needed for generating problems using NETGEN. Problems 9-14 are transportation problems and the rest are transshipment problems. The next set of problems is generated with the intent of having problems with comparatively higher costs on a greater percentage of arcs. The dimensions and the capacity ranges in this set are same as in the first, but the cost range is 1-100, the percentage of the capacitated arcs is fixed at 50 and the percentage of high cost arcs is 30 or 80.

Computational Experience

The Sequent Symmetry S81 has been used for all computational studies. The times reported are wall clock times in seconds and do not include the time taken for input of the data or the output of the results. Table 1 lists the time taken by FW, PWFW and PWLAG to solve 20 problems, to a required accuracy of 99%, generated with the above specifications.

The time ratio in the table gives the time taken by PWFW or PWLAG as a fraction of that taken by FW for the same problem. PWLAG can obtain solutions with the desired accuracy in 2%-53% of the FW time, depending on the problem size. PWFW takes 5%-77% of the FW time. The computational advantage of PWLAG or PWFW over FW is seen to be extremely substantial as the problem size increases. On the average, PWFW takes 36 percent of the time taken by FW and PWLAG takes 24 percent. We note that the computational advantage of PWFW and PWLAG over FW increases with the problem size. For problems 15-20, which are transshipment problems with 500-600 nodes and 8000-15,000 arcs, PWFW terminates with an ϵ-optimal solution in only 5%-11% of the FW time. PWLAG performs even better and requires only 2%-5% percent of the time taken by FW. From our computational experience the improvement in time ratios with problem size is seen to be more pronounced for transshipment problems as compared to that for transportation problems. Figures 1-3 are graphs of the time taken by the three algorithms for problems 1-6, 7-14 and 15-20 respectively and clearly demonstrate the performance superiority of PWFW and PWLAG over FW.

PWLAG takes less time than PWFW in general, and this is mainly attributed to the fact that the quality of the bound obtained through the Lagrangian relaxation is better, that is, the bound is tighter, and therefore, the termination criterion is satisfied in a fewer number of iterations. If the final objective function values obtained by PWFW and PWLAG are compared, the former is found to terminate with a lower value in general. The fact that the bound in PWLAG requires less computational time than the bound in PWFW also contributes to the time savings observed. Specially, for quadratic cost functions, calculation of the Lagrangian bound is much cheaper than obtaining the Frank-Wolfe bound. The latter requires the solution of a network flow problem. On the average, the percentage of time required to calculate the Frank-Wolfe bound by PWFW is seen to be nearly double of that required to obtain the Lagrangian bound by PWLAG. This may or may not be the case for general convex cost problems.

We also determined the percentage of time that PWFW and PWLAG expend in the refinement process and on the average it was found to be comparable for the two algorithms.

Since PWFW and PWLAG are integer implementations while FW is a *real* FORTRAN code, we developed real versions of PWFW and PWLAG called PWFWR and PWLAGR, respectively, for the purpose of comparison. Table 2 lists the times taken by these versions to solve the original set of problems to a minimum accuracy of 99%. As expected, the real versions do not have the same computational advantage over FW as the integer implementations do. On the average the time taken by PWFWR for the 20 problems is 88 percent of the FW time and that taken by PWLAGR is 55 percent. The real versions do not always perform better than FW but for the large problems the time savings are substantial. PWLAGR takes 7-14 percent of the FW time on problems 15-20 and the time taken by PWFWR is 11-27 percent. It should be noted that since the integer implementations can be used to obtain solutions with the desired accuracy of 99%, the real versions are developed for the purpose of comparative study alone and need not be used.

The requirement that the objective function value be at least 99% accurate at termination is quite stringent, specially since the FW method is known to perform poorly as the solution approaches the optimal. The accuracy requirement is therefore relaxed to 95% and the problem set is tested again. The times taken by the 3 algorithms are tabulated in Table 3. As expected the computational advantage of PWFW and PWLAG for this level of accuracy is slightly less than before. The average time taken by PWFW is 47% of the FW time and that taken by PWLAG is 30% which is again a considerable amount of computational advantage.

Table 4 lists the times taken by FW, PWFW and PWLAG to solve the 20 problems in the second set. The odd and even numbered problems in this set have 30% and 80% high cost arcs respectively. These problems were generated to gauge the effect of increasing the cost range and the percentage of high cost arcs on the time performance. From the results in Table 4 we conclude that the time savings on the two algorithms for this set of problems are quite similar to that for the first set of problems, with PWFW taking 40% and PWLAG taking 25% of the FW time, and therefore the cost structure does not have a considerable effect on their performance. It should be noted though that in two cases PWFW terminates with a solution which is guaranteed to be above 98% optimal but does not meet the requirement of 99% accuracy. Iteration count is provided as an optional termination cr'terion for cases where the desired accuracy is not met within 15 iterations, where an iteration consists of obtaining the solution to the current approximation and evaluating the upper and lower bounds. The upper limit on the number of iterations is chosen to be 15 since it is observed that the solution does not show substantial improvement thereafter in most cases.

5. Other Strategies and Modifications

In most cases it was observed that the values of the lower and upper bounds improved at each successive iteration. This is not guaranteed by our approach. We, therefore, implemented a strategy based on Meyer's [24] paper that guarantees a monotonically decreasing objective function value of (P) at each iteration. This requires that the optimal solution to $\tilde{P}_k$ at the i^{th} iteration be a breakpoint in the next iteration for all basic and nonbasic arcs. Previously this was enforced at each refinement for the nonbasic arcs alone and in fact, it was essential to do so in order to satisfy the definition of a nonbasic arc. With this adaptation (PWMY), the solution does improve at each iteration, but the strategy does not achieve any computational advantage over the previous one, especially since the desired accuracy is not achieved for half of the problems in the set. An overall comparison of the times taken by PWMY, PWFW, PWLAG and FW is given in Table 5. It is to be noted that the time listed for PWMY is the time taken for obtaining the solution with the indicated accuracy achieved rather than the desired accuracy.

6. Summary and Further Research

The current computational experience establishes the effectiveness of our piecewise linear approach for solving quadratic convex cost problems. It has a considerable advantage over an implementation of the Frank-Wolfe method. Large problems can be solved to within 99% of the optimal solution in as little as 4% of the FW time. On an average PWLAG takes 24 percent of the time taken by FW for the set of 20 randomly generated problems that were tested.

In the absence of standard benchmark problems of the form $(P)$with known optimal solutions and objective function values, our present study suffices to successfully demonstrate the computational superiority of our approach over a traditional method such as the Frank-Wolfe.

A natural extension of the work would be a solution procedure for nonseparable convex cost problems. We are also interested in investigating if an enhancement of our implementation could obtain at least locally optimal solutions for concave cost problems, which have extensive applications but unfortunately are difficult problems to solve.

Table I TIME‡ TAKEN FOR ORIGINAL PROBLEMS

PRB NUM	FW	PWFW	TIME RATIO*	PWLAG	TIME RATIO*	OPTIMAL VALUE
1	1.64	.94	.57	.48	.29	-8592216
2	1.76	1.11	.63	.97	.55	-5766709
3	8.66	2.71	.31	1.54	.18	-76484447
4	7.63	3.38	.44	2.25	.29	-54936770
5	11.21	3.77	.34	2.67	.24	-40527843
6	8.41	4.16	.50	2.88	.34	-45898496
7	56.06	11.89	.21	6.39	.11	-261093554
8	38.16	20.74	.54	13.30	.35	-208477916
9	29.93	17.55	.59	12.65	.42	-116588999
10	34.53	14.38	.42	10.13	.29	-129284946
11	119.16	44.47	.37	32.99	.28	-72016847
12	99.66	37.22	.37	27.09	.27	-76165575
13	145.93	109.41	.75	73.47	.50	-64705809
14	206.42	106.87	.52	74.61	.36	-52456271
15	1502.02	129.52	.09	78.07	.05	-589931746
16	1163.17	121.03	.10	61.08	.05	-533283869
17	2198.61	119.69	.05	69.87	.03	-572189947
18	1546.18	95.56	.06	53.72	.03	-489618838
19	3700.19	177.23	.05	105.41	.03	-476943806
20	2841.69	135.55	.05	69.88	.02	-440758194
		Average Ratio	.35		.24	

‡Wall clock seconds on the Sequent Symmetry S81 *With respect to FW

Table II TIME[‡] TAKEN FOR ORIGINAL PROBLEMS ON REAL VERSIONS

PRB NUM	FW	PWFWR	TIME RATIO*	PWLAGR	TIME RATIO*	OPTIMAL VALUE
1	1.65	2.42	1.47	1.36	.83	-8592216
2	1.75	2.50	1.43	2.19	1.25	-5766709
3	8.68	5.25	.61	3.20	.37	-76484447
4	7.62	11.94	1.57	5.01	.66	-54936770
5	11.24	10.23	.91	5.71	.51	-40527843
6	8.34	12.60	1.51	6.49	.78	-45898496
7	55.98	31.07	.55	13.83	.25	-261093554
8	38.65	53.12	1.37	44.63	1.15	-208477916
9	29.51	41.26	1.40	28.17	.95	-116588999
10	34.46	34.38	1.00	22.41	.65	-129284946
11	118.79	107.17	.90	74.30	.63	-72016847
12	99.71	105.97	1.06	58.61	.59	-76165575
13	145.69	231.61	1.59	167.62	1.15	-64705809
14	206.44	263.75	1.28	147.75	.72	-52456271
15	1500.35	408.55	.27	165.85	.11	-589931746
16	1167.24	308.73	.26	129.52	.11	-533283869
17	2199.81	320.64	.15	152.70	.07	-572189947
18	1542.60	210.47	.14	112.04	.07	-489618838
19	3689.05	420.14	.11	222.94	.06	-476943806
20	2834.04	313.88	.11	139.84	.05	-440758194
		Average Ratio	.88		.55	

[‡]Wall clock seconds on the Sequent Symmetry S81 *With respect to FW

Table III TIME[‡] TAKEN FOR ORIGINAL PROBLEMS WITH 95% ACCURACY

PRB NUM	FW	PWFW	TIME RATIO*	PWLAG	TIME RATIO*	OPTIMAL VALUE
1	1.21	.76	.62	.37	.30	-8592216
2	1.13	1.00	.88	.58	.51	-5766709
3	6.32	2.13	.34	1.11	.18	-76484447
4	4.72	2.42	.51	1.73	.37	-54936770
5	7.76	2.76	.36	1.90	.24	-40527843
6	5.35	4.17	.78	2.30	.43	-45898496
7	39.99	10.42	.26	5.62	.14	-261093554
8	24.06	16.91	.70	8.44	.35	-208477916
9	16.52	14.49	.88	9.34	.57	-116588999
10	17.47	11.54	.66	8.05	.46	-129284946
11	63.79	35.92	.56	24.85	.39	-72016847
12	53.78	27.66	.51	23.04	.43	-76165575
13	82.67	76.62	.93	65.34	.79	-64705809
14	107.34	86.86	.81	53.06	.49	-52456271
15	1107.08	111.28	.10	67.92	.06	-589931746
16	806.97	96.14	.12	39.34	.05	-533283869
17	1534.34	121.18	.08	58.28	.04	-572189947
18	1051.52	78.72	.07	45.18	.04	-489618838
19	2441.02	155.65	.06	87.61	.04	-476943806
20	1815.70	124.83	.07	58.04	.03	-440758194
		Average Ratio	.47		.30	

[‡]Wall clock seconds on the Sequent Symmetry S81 *With respect to FW

Table IV TIME‡ TAKEN FOR PROBLEMS WITH HIGHER COSTS

PRB NUM	FW	PWFW	ACCUR-ACY(%)	TIME RATIO+	PWLAG	TIME RATIO+	OPTIMAL VALUE
1	3.14	1.46	99.5	.46	.89	.28	-18802914
2	1.94	.97	99.3	.50	.71	.37	-29763764
3	7.77	3.95	99.2	.51	2.34	.30	-143248991
4	6.59	3.25	99.0	.49	2.01	.30	-149300640
5	8.63	4.88	99.5	.57	2.95	.34	-97459866
6	8.87	5.27	99.3	.59	3.38	.38	-103358663
7	44.58	19.32	99.1	.43	9.53	.21	-480888274
8	45.90	22.11	98.5	.48	13.62	.30	-357156520
9	40.54	20.55	99.7	.51	12.17	.30	-190760881
10	33.28	20.24	99.8	.61	12.59	.38	-212969924
11	100.28	55.18	99.2	.55	36.70	.37	-113307805
12	111.40	52.14	99.1	.47	35.88	.32	-124435503
13	191.76	109.81	99.0	.57	76.47	.40	-84864094
14	185.58	128.95	99.6	.69	82.89	.45	-94472368
15	1584.02	141.67	99.3	.09	88.48	.06	-538275687
16	1299.63	138.23	98.8	.11	70.08	.05	-765287147
17	1916.40	123.30	99.4	.06	67.56	.04	-809483905
18	2111.48	126.92	99.4	.06	73.18	.03	-711384195
19	3522.58	170.65	99.3	.05	102.27	.03	-606614724
20	3731.50	156.23	99.7	.04	92.43	.02	-554617067
			Average Ratio	.39		.25	

‡Wall clock seconds on the Sequent Symmetry S81 +With respect to FW

*Accuracy achieved less than 99%

Table V TIME‡ TAKEN FOR ORIGINAL PROBLEMS

PRB NUM	FW	PWFW	TIME RATIO+	PWLAG	TIME RATIO+	PWMY	ACCUR-ACY(%)	TIME RATIO+
1	1.64	.94	.57	.48	.29	.75	99.2	.46
2	1.76	1.11	.63	.97	.55	1.06	99.1	.60
3	8.66	2.71	.31	1.54	.18	2.80	99.2	.32
4	7.63	3.38	.44	2.25	.29	4.44	99.4	.58
5	11.21	3.77	.34	2.67	.24	4.09	99.3	.36
6	8.41	4.16	.50	2.88	.34	4.36	99.3	.52
7	56.06	11.89	.21	6.39	.11	26.32	98.5*	.47
8	38.16	20.74	.54	13.30	.35	23.77	97.0*	.62
9	29.93	17.55	.59	12.65	.42	19.07	99.0	.64
10	34.53	14.38	.42	10.13	.29	36.78	98.9*	1.07
11	119.16	44.47	.37	32.99	.28	69.40	98.5*	.58
12	99.66	37.22	.37	27.09	.27	79.26	98.8*	.80
13	145.93	109.41	.75	73.47	.50	172.01	98.6*	1.18
14	206.42	106.87	.52	74.61	.36	205.86	98.8*	1.00
15	1502.02	129.52	.09	78.07	.05	363.17	98.8*	.24
16	1163.17	121.03	.10	61.08	.05	194.60	99.2	.17
17	2198.61	119.69	.05	69.87	.03	260.11	98.3*	.12
18	1546.18	95.56	.06	53.72	.03	200.09	99.3	.13
19	3700.19	177.23	.05	105.41	.03	366.14	91.7*	.10
20	2841.69	135.55	.05	69.88	.02	235.53	99.2	.08
		Average Ratio	.35		.24			.50

‡Wall clock seconds on the Sequent Symmetry S81 +With respect to FW

*Accuracy achieved less than 99%

7. References

[1] Bachem, A. and B. Korte, *Minimum Norm Problems Over Transportation Polytopes,* Linear Algebra and its Applications, Vol. 31, pp. 103-118, 1980.

[2] Beck, P., L. Lasdon and M. Engquist, *A Reduced Gradient Algorithm for Nonlinear Network Flow Problems,* ACM Trasactions on Mathematical Software, Vol. 9, pp. 57-70, 1983.

[3] Beckmann, M., C.B. McGuire and C. Winsten, *Studies in Economics of Transportation,* Yale University Press, New Haven, Conn., 1956.

[4] Collins, M., L. Cooper, R. Helgason, J. Kennington and L.J. LeBlanc, *Solving the Pipe Network Analysis Problem using Optimization Techniques,* Management Science, Vol. 24, pp. 747-760, 1978.

[5] Bertsekas. D.P. and D. El Baz, *Distributed Asynchronous Relaxation Methods for Convex Network Flow Problems.* SIAM Journal of Control and Optimization, Vol. 25, pp. 74-85, 1987.

[6] Bertsekas. D.P., P.A. Hosein and P. Tseng, *Relaxation Methods for Network Flow Problems with Convex Arc Costs,* SIAM Journal of Control and Optimization, Vol. 25, pp. 1219-1243, 1987.

[7] Cooper, L. and J. Kennington, *Steady State Analysis of Nonlinear Resistive Electrical Networks Using Optimization Techniques.* Technical Report IEOR 77012 Southern Methodist University. Dallas. TX. 1977.

[8] Cooper. L. and L.J. LeBlanc. *Stochastic Tranportation Problems and other Network Related Convex Problems.* Naval Research Logistics Quarterly, Vol. 24. No. 2. 1977.

[9] Dantzig. G.B.. *Linear Programming and Extensions.* Princeton University Press. Princeton. NJ. 1963.

[10] Dembo, R.S.. *A Primal Truncated Newton Algorithm with Application to Large-Scale Nonlinear Network Optimization,* Mathematical Programming Study. Vol. 31, pp. 43-72. 1897.

[11] Dembo. R.S. and J.G. Klincewicz. *A Scaled Reduced Gradient Algorithm for Network Flow Problems with Convex Separable Costs,* Mathematical Programming Study. Vol. 15, pp. 125-147. 1981.

[12] Glassey. C.R., *A Quadratic Network Optimization Model for Equilibrium of Single Commodity Trade Flows,* Mathematical Programming , Vol. 14, pp. 98-107. 1978.

[13] Geoffrion, A.M., *Objective Function Approximation in Mathematical Programming,* Mathematical Programming . Vol. 13, pp. 23-27, 1977.

[14] Hadley. G., *Nonlinear and Dynamic Programming,* Addison-Wesley. Reading. MA. 1964.

[15] Hearn. D.W.. S. Lawphongpanich and J.A. Ventura, *Restricted Simplicial Decomposition: Computations and Extensions.* Mathematical Programming Study. Vol. 31. pp. 99-118. 1987.

[16] Helgason. R.V. and J.L. Kennington. *An Efficient Specialization of the Convex Simplex Method for Nonlinear Network Flow.* Technical Report No. IEOR 77017. Dept. of Computer Science and

Engineering, Southern Methodist University, Dallas, TX, 1978.

[17] Kao, C.Y. and R.R. Meyer, *Secant Approximation Methods for Convex Optimization*, Mathematical Programming Study, Vol. 14, pp. 143-162, 1981.

[18] Kennington, J.L. and R.V. Helgason, *Algorithms for Network Programming,* Wiley-Interscience, NY, 1980.

[19] Klincewicz, G.J., *A Newton Method for Convex Separable Network Flow Problems,* Networks. Vol. 13, pp. 427-442, 1983.

[20] Klincewicz, G.J., *Implementing an 'Exact' Newton Method for Separable Convex Transportation Problems,* Networks, Vol. 19, 1989.

[21] Klingman, D., A. Napier and J. Stutz, *NETGEN: A Program for Generating Large Scale Capacitated Assignment. Transportation, and Minimum Cost Flow Network Problems,* Management Science, Vol. 20. N0. 5. pp. 814-821. 1974.

[22] LeBlanc, L.J.. *The Conjugate Gradient Technique for Certain Quadratic Network Problems.* Naval Research Logistics Quarterly. Vol. 23. pp. 597-602. 1976.

[23] LeBlanc, L.J.. R.V. Helgason and D.E. Boyce, *Improved Efficiency of the Frank-Wolfe Algorithm for Convex Network Programs.* Transportation Science. Vol. 19, 445-462. 1985.

[24] Meyer. R.R., *Two-Segment Separable Programming,* Management Science, Vol. 23. No. 4. pp. 385-395. 1979.

[25] Meyer. R.R.. *Computational Aspects of Two-Segment Separable Programming.* Mathematical Programming . Vol. 26. pp. 21-39. 1983.

[26] Minoux. M.. *A Polynomial Algorithm for Minimum Quadratic Cost Flow Problems.* European Journal of Operational Research. Vol. 18. pp. 377-387. 1984.

[27] Murthy. R.V. and Helgason R.V., *A Direct Simplex Algorithm for Network Flow Problems with Piecewise Linear Costs,* Technical Report No. 93-CSE-7, Dept. of Computer Science and Engineering. Southern Methodist University, Dallas, TX. 1993.

[28] Ouchi A. and I. Kaji. *Lagrangian Dual Coordinatewise Maximization Algorithm for Network Transportation Problems with Quadratic Costs,* Networks, Vol. 14, pp. 515-530, 1984.

[29] Ravindran. A. and H.K. Lee. *Computer Experiments on Quadratic Programming Algorithms.* European Journal of Operational Research, Vol. 8, pp. 166-174. 1981.

[30] Saaty. T.L., *Optimization in Integer and Related Extremal Problems.* McGraw-Hill. NY. 1970.

[31] Thakur. L.S.. *Error Analysis for Convex Separable Programs: The Piecewise Linear Approximation and the Bounds on the Objective Function Value.* SIAM Journal on Applied Mathematics. Vol. 34: pp. 704-714. 1978.

[32] Wagner, H.M., *Principles of Operations Research with Applications to Managerial Decisions*, Prentice-Hall, Englewood Cliffs, NJ, 1969.

[33] Wets, R., *Programming Under Uncertainity: The equivalent Convex Program*, SIAM Journal on Applied Mathematics, Vol. 14, pp. 89-105, 1966.

[34] Zenios, S.A. and J.M. Mulvey, *Relaxation Techniques for Strictly Convex Network Problems*, Annals of Operations Research, Vol. 5, pp. 517-538, 1985/6.

Graphs of the Times Taken with Increasing Number of Segments

Fig 1 SOLUTION TIMES(1-6)

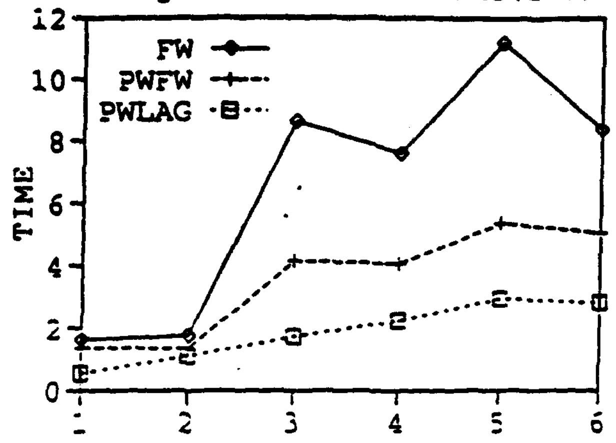


Fig 2 SOLUTION TIMES(7-14)

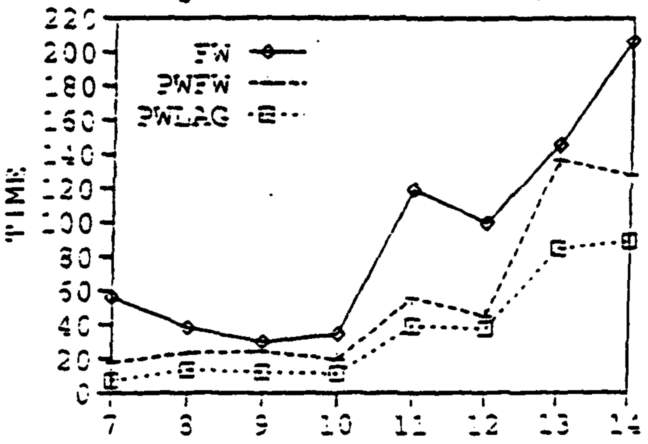
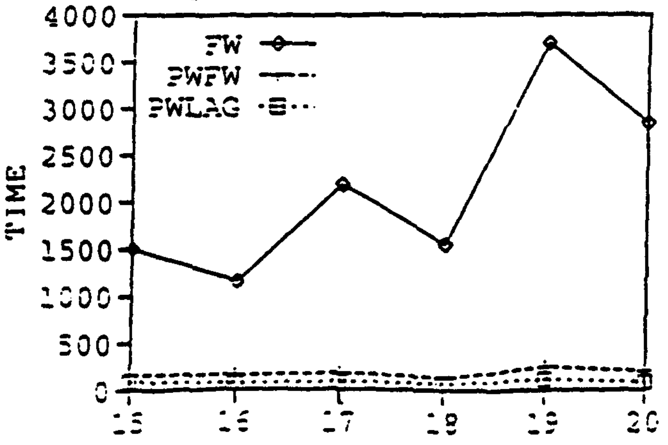


Fig 3 SOLUTION TIMES(15-20)



Computational Study of Implementational Strategies For The Network Penalty Method

Nandagopal Venugopal and R. V. Helgason
Department of Computer Science and Engineering,
Southern Methodist University, Dallas, Texas-75205

September 20, 1993

Abstract

In this paper we present two implementations of the network penalty method as proposed by Conn, Gamble and Pulleyblank. The results of our computational testing and comparative study with the network simplex method are presented.

1 Introduction:

The simplex method for solving linear programs has been effectively adapted to solve minimum cost network flow problems, wherein the special network structure can be exploited. The network simplex method [1] is still one of the most popular techniques for solving these problems. Gamble, Conn and Pulleyblank [2] suggested a penalty algorithm for solving the minimum cost network flow problem. This algorithm bears a strong resemblance to the network simplex method and is a specialization of the Conn [3, 4] and Bartels' [5] nonlinear penalty function methods. The penalty method, unlike the network simplex, permits infeasible flows, i. e. the flows in the arcs are not required to be within the bounds. Infeasible flows incur a nonnegative penalty in the objective function and this encourages the flows back towards feasibility. In this version of the penalty algorithm (as in [2]) only the basic flows are permitted to be infeasible and the nonbasic flows are required to be at bound. It should be noted that this is only for convenience and the same idea could be extended to treat infeasible nonbasic flows.

We begin by introducing the notations being followed in our presentation. Let $\mathbf{G(m,n)}$ be a connected, directed graph with $\mathbf{m}$ nodes and $\mathbf{n}$ arcs. Let $\mathbf{A}$ be the node arc incidence matrix for the graph $\mathbf{G}$, where each row of $\mathbf{A}$ represents a node in $\mathbf{G}$ and each column represents an arc in $\mathbf{G}$. In order to ensure that the matrix $\mathbf{A}$ has full row rank, a column corresponding to a *root arc* is appended. The node r on which the root arc is incident is called the *root node*. Each element a_{ij} of $\mathbf{A}$ is defined by:

$$a_{ij} = \begin{cases} +1 & \text{if i is the ``from'' node (tail) of arc j} \\ -1 & \text{if i is the ``to'' node (head) of arc j} \\ 0 & \text{otherwise} \end{cases}$$

For every arc e_j, let $F(e_j)$ denote its *from* node and $T(e_j)$ its *to* node. Let $\mathbf{c}$ be the vector of arc costs, $\mathbf{u}$ the upperbounds on the arc flows, and $\boldsymbol{x}$ the actual flows. Let the node requirements be given by the vector $\mathbf{b}$. The minimum cost network flow problem in standard form can then be defined as:

$$\begin{array}{rl} \textit{minimize} & \boldsymbol{cx} \\ \textit{subject to} & \boldsymbol{Ax} = \boldsymbol{b} \\ & \mathbf{o} \leq \boldsymbol{x} \leq \boldsymbol{u} \end{array}$$

The matrix $\mathbf{A}$ can be partitioned into $[\mathbf{B}|\mathbf{N}]$, where $\mathbf{B}$ is the basis and $\mathbf{N}$ is the set of nonbasic arcs (which are at a bound). In the graph $\mathbf{G}$, we define $\mathbf{T}_B$ to be the rooted spanning tree corresponding to the basis $\mathbf{B}$.

In the penalty algorithm, the basic arcs are permitted to have infeasible flows and as such we consider the basis $\mathbf{B}$ to be partitioned as $[\mathbf{F}|\mathbf{M}|\mathbf{P}]$ where $\mathbf{F}$ represents the basic arcs which carry feasible flows, $\mathbf{M}$ the set of basic arcs which have violations of their lower bounds (or negative flows) and $\mathbf{P}$, consisting of basic arcs with flows exceeding their upperbounds. Each unit of infeasibility incurs a penalty α in the objective function. We can thus express the penalty problem as:

$$\begin{array}{ll} \textit{minimize} & \boldsymbol{\Phi}_\alpha(\boldsymbol{x}) \\ \textit{subject to} & \boldsymbol{Ax} = \boldsymbol{b} \\ & \boldsymbol{x}_B \text{ unrestricted} \\ & \mathbf{o} \leq \boldsymbol{x}_N \leq \boldsymbol{u}_N \end{array}$$

where $\boldsymbol{\Phi}_\alpha(\boldsymbol{x})$ is given by:

$$\boldsymbol{\Phi}_\alpha(\boldsymbol{x}) = \mathbf{c}_N\boldsymbol{x}_N + \sum_{j\in F}\mathbf{c}_j\boldsymbol{x}_j + \sum_{j\in M}(\mathbf{c}_j - \boldsymbol{\alpha})\boldsymbol{x}_j + \sum_{j\in P}(\mathbf{c}_j + \boldsymbol{\alpha})\boldsymbol{x}_j$$

Thus for a given set of basic flows, the current objective function coefficients (penalty function coefficients) can be computed from the following relationship:

$$c'_j = \begin{cases} c_j & \text{if } j \in (F \cup N) \\ c_j - \alpha & \text{if } j \in M \\ c_j + \alpha & \text{if } j \in P \end{cases}$$

We define the vector $\boldsymbol{\pi}$ to be the usual network simplex duals on $\mathbf{T}_B$, calculated using only the original costs on the basic arcs. Let the vector $\boldsymbol{\pi}'$ denote the actual (correct) duals, i.e. , the duals that reflect the penalty associated with any infeasible basic arcs in $\mathbf{T}_B$. The duals $\boldsymbol{\pi}'$ are calculated using the penalty function coefficients $\mathbf{c}'$. As in the network simplex, if an improving direction is found in the pricing step, then by adding this improving nonbasic arc to the spanning tree $\mathbf{T_B}$ a cycle is created. Flow is augmented on such a cycle until $\boldsymbol{\Phi}_\alpha(\boldsymbol{x})$ is no longer improving. This occurs when the flow augmentation is such that the leaving arc reaches a bound when $\lceil \overline{\mathbf{c}_j}'/\alpha \rceil + 1$ bounds have been crossed on the cycle, where $\overline{\mathbf{c}_j}'$ is the reduced cost of the entering arc e_j.

The paper is organized as follows. In section 2 we describe our implementations of the network penalty algorithm of Conn, Gamble, and Pulleyblank [2]. Section 3 discusses our first implementation—a dual updating technique. In section 4 we present a cycle tracing technique for pricing used in our second implementation. Section 5 discusses the problem set used in the computational testing of our implementations. In section 6 we present the computational results and conclusions.

2 Network penalty algorithm implementation:

Before presenting our implementations, it is necessary to describe, briefly, the method for choosing the leaving variable. This is the same in both our implementations. Recall that the basis $\mathbf{B}$ has a corresponding rooted spanning tree $\mathbf{T_B}$ in $\mathbf{G}$. Let e_j be the entering arc. Adding this arc to $\mathbf{T_B}$ creates a cycle. We define the *forward cycle* $FC(e_j)$ as $F(e_j)$, e_j, followed by the unique path in $\mathbf{T_B}$ connecting $T(e_j)$ and $F(e_j)$. An arc $e_k \in FC(e_j)$ is called a *forward arc* if, in the path from $T(e_j)$ to $F(e_j)$, its orientation is $\{F(e_k), e_k, T(e_k)\}$. It is said to be a *reverse arc* otherwise. The reverse of $FC(e_j)$, denoted by $RC(e_j)$, is called the *reverse cycle* of e_j and its forward and reverse arcs can be defined in a similar manner. We define a node k of the cycle $FC(e_j)[RC(e_j)]$ to be the *common node* for the cycle if it is common to the paths connecting $F(e_j)$ and $T(e_j)$ to the root node r. If k is the common node for the above cycles then k is also the node in the cycle that is the closest to the root node, where the distance of a node from the root is measured by the number of arcs in the path connecting it to the root.

Suppose the arc e_j is at lowerbound [upperbound]. Then flow will be increased on $FC(e_j)[RC(e_j)]$. Forward arcs in $FC(e_j)[RC(e_j)]$ that have feasible flows, or infeasible flows due to lowerbound violations are likely candidates to leave. In the first case they could leave upon attaining their upperbound, while in the second case they also could have the option of leaving at the lowerbound. Similarly, reverse arcs in $FC(e_j)[RC(e_j)]$ which will have flow decreased could be candidates to leave if they are feasible and the flow change reduces the flow in them to the lowerbound, or if they are infeasible due to upperbound violations and the flow change results in their attaining either of their bounds. In any event, all such arcs in which a nonnegative change could result in their attaining a bound are candidates for leaving the basis. All possible nonnegative flow changes that could create a bound crossing are sorted in ascending order. The arc corresponding to the $\lceil \overline{c_j}'/\alpha \rceil$th flow change is chosen to be the leaving variable in [2]. However, in our implementation we pick the arc corresponding to the $\lfloor \overline{c_j}'/\alpha \rfloor$th flow change to be the leaving arc. Thus in our implementation we restrict the number of violations that can occur to be strictly within the improving range for $\Phi_\alpha(x)$. We now present the pseudocode for our implementation of the network penalty algorithm. We have two distinct methods for handling the computation of correct duals, the ***dual updating*** technique and the *cycle tracing* technique, hereafter referred to as **DU** and **CT** respectively. Either can be used in the scheme described below. At points where they differ we describe them explicitly.

Network penalty algorithm:

```
Step 1:
    Let $\mathbf{B} = [\mathbf{F}|\mathbf{M}|\mathbf{P}]$ be any basis. Let $\boldsymbol{x}$ be the initial primal solution.
    Note that $\boldsymbol{x}$ need not satisfy the arc bounds.
    Choose some $\alpha > 0$.

Step 2:
    DU:
    Compute the current penalty function coefficients $\mathbf{c}'$ (as discussed in
    section 1).
    CT:
    Here we use the original cost vector $\mathbf{c}$

Step 3:
    DU:
    Calculate the dual solution $\boldsymbol{\pi}'$ using the penalty coefficients $\mathbf{c}'$.
    CT:
    Calculate the normal duals $\boldsymbol{\pi}$ using the original arc costs $\mathbf{c}$.

Step 4:  Select the entering variable.

    DU:
    $\forall\ e_j \in \mathbf{N} = [\mathbf{L}|\mathbf{U}]$ compute $\overline{c_j}' = c_j - \pi'_{F_{[e_j]}} + \pi'_{T_{[e_j]}}$
    Choose an arc $e_j \in \mathbf{U} \ni \overline{c_j}' > 0$, $e_j \in \mathbf{L} \ni \overline{c_j}' < 0$;
    providing an improving direction.
    If no such arc exists, go to Step 10.
    CT:
    Using suitable selection crietria select a possible candidate arc
    $e_j$, to enter the basis.
```

Determine the correct reduced cost $\overline{c_j}'$ by tracing the cycle created by adding e_j to $\mathbf{T}_B$.
As in the **DU** implementation, choose an arc e_j if it provides an improving direction.
Else go to Step 10.

Step 5:
Adding e_j to $\boldsymbol{T}_B$ creates a cycle (with two possible orientations).
Let the working cycle $C(e_j)$ be defined by:

$$C(e_j) = \begin{cases} FC(e_j) & if\ e \in \mathbf{L}, \\ RC(e_j) & if\ e \in \mathbf{U}. \end{cases}$$

Step 6: Determine the leaving variable and amount of flow augmentation on the cycle.

Let $\Gamma_f = \{\Phi\}$
Let $\Gamma_r = \{\Phi\}$
For every forward arc e_k in $C(e_j)$, let

$$\Gamma_f = \Gamma_f \ \cup \begin{cases} \{u_k - x_k\} & if\ e_k \in F, \\ \{u_k - x_k, -x_k\} & if\ e_k \in M. \end{cases}$$

For every backward arc e_k in $C(e_j)$, let

$$\Gamma_r = \Gamma_r \ \cup \begin{cases} \{x_k\} & if\ e_k \in F, \\ \{x_k - u_k, x_k\} & if\ e_k \in P. \end{cases}$$

$\Gamma = \Gamma_f \ \cup \ \Gamma_r$

Step 7:
Let $p = \lfloor \overline{c_j}'/\alpha \rfloor$
If $p < |\Gamma|$, then the problem is unbounded for the current value of α.
In [2] it is suggested that the algorithm stop at this point.
However, in both our implementations, instead of stopping, we perform a simplex type pivot.

Step 8:
Pick the arc corresponding to the p^{th} smallest element in Γ to be the leaving arc.
Let Δ be the p^{th} smallest element in Γ, the amount of flow augmentation on $C(e_j)$.

Step 9: Perform the pivot operation
Update $\mathbf{x}$ by augmenting Δ units of flow on $C(e_j)$.
(For every forward arc in $C(e_j)$ add Δ units of flow and for every backward arc subtract Δ units of flow.)
Pivot in the entering arc and delete the leaving arc from $\mathbf{B}$.
DU:
Using the *Dual Updating* scheme, update $\mathbf{B}$.
Go to Step 4.
CT:
Carry out a *normal dual* update on only the modified portion of the tree.
Go to Step 4.

Step 10: Stopping Condition.
If the flows $\mathbf{x}$ are feasible then stop with an optimal solution.
Else, increase α and return to Step 1.

The two implementations presented above, differs principally in the manner in which the duals are computed. This in turn affects the manner of pricing the nonbasic arcs, determining a suitable candidate arc to enter the basis. Consider an arc e_j which is a candidate to enter the basis. In the *dual updating* technique, since the correct duals π' are maintained, the reduced cost $\overline{c}_j'$ for the nonbasic arc computed using these duals would reflect the infeasibilities, if any, present in the basis. Thus the choice of the entering arc can be made on the basis of its reduced cost computed directly from the duals. On the other hand, in the *cycle tracing* technique, the duals π, computed using only the original arc costs, do not reflect penalties for any infeasible flows present in the basis. Thus, in selecting an arc to enter the basis, we first compute its reduced cost, $\overline{c}_j$ using π, and then correct it by doing a trace of the cycle formed by adding it to the basis. This cycle trace is done to determine the effect of any infeasible flows present in the cycle on the reduced cost of the entering arc. Thus the correct reduced cost $\overline{c_j'}$ is available only after doing the cycle trace. The decision on whether to let e_j enter the basis is now made based on $\overline{c_j'}$.

Network problems are highly degenerate. It is therefore quite possible for a basic arc to be at a bound. Depending on the direction of flow augmentation on the cycle, it is possible for these degenerate arc flows to have two costs. But we restrict such arc flows to have just one cost, equal to the original cost, and if the flow augmentation is such that the arc flow becomes infeasible, we will then treat it as if it were a blocking arc that reached a bound immediately. We thus permit the algorithm to make degenerate pivots. This is necessary, since otherwise the algorithm could terminate with a suboptimal solution.

3 Dual updating technique:

As mentioned previously, with this technique we maintain the correct duals. We initiate the algorithm with any basis **B** and flows **x**. The basic flows $\mathbf{x}_B$ need not be feasible, but the nonbasic flows $\mathbf{x}_N$ are assumed to be at bound. For every basic arc e_j, its correct cost c_j' is computed (as in section 1). The dual π_r' at the root node r, is set equal to 0. To obtain the other dual values we do a depth-first traversal of the tree from r, setting the reduced cost $\overline{c_j'}$ of each arc $e_j \in \mathbf{B}$ to zero. Thus, knowing the dual at one of the nodes for an arc, the dual at the other node is immediate by addition or subtraction.

For every node k in the tree we maintain a small amount of node-label information. Level(k) gives the height of the node in the tree. Note that level(r) is 0. The height is the distance of the node from the root r. Next(k) gives the following node in a thread which provides a particular depth first traversal of the tree. Arcid(k) gives the index of the arc in the tree that is connected to k and is on the path to the root. Finally, Down(k) is the preceding (predecessor) node in the path connecting k to the root. The arcid at a node points to input information about the arc such as its from node, cost and upperbound. Using the data structure described above we now present details of the *dual updating* implementation for step 3 of the network penalty algorithm.

Step 3 — DU: (Computing the duals on the tree)

```
$\pi'_r \leftarrow 0$
n1 $\leftarrow r$
n2 $\leftarrow$ next(n1)
while (height(n2) > 0) do:
    $e_j \leftarrow$ arcid(n2)
    $c'_j \leftarrow$ current penalty coefficient for arc $e_j$ (see section 1)
    $\pi'_{n2} = \begin{cases} \pi'_{n1} + c'_j & if\ n2 = F(e_j) \\ \pi'_{n1} - c'_j & if\ n2 = T(e_j) \end{cases}$
    n2 $\leftarrow$ next(n2)
    n1 $\leftarrow$ down(n2)
enddo
```

Steps 4 – 8 of the network penalty algorithm describe how to select a suitable arc e_j to enter the basis, how much flow Δ is to be augmented on the cycle $C(e_j)$, and try to determine the arc in the cycle that is to leave the basis. The *pivot operation in step* 9 affects the duals in the tree in two possible ways.

When flow is augmented on the cycle, it is possible that arcs which were initially feasible now violate their bounds and vice versa. We describe this as a change in the *character* of the arc. When this happens, the arc cost before and after the pivot is different. Thus nodes attached to the tree through this arc, after the pivot, will have changed dual values. When the leaving arc is deleted from the tree, the tree becomes disconnected. The nodes detached by the above operation are reattached to the tree (or *rehung*) through the new entering arc. The duals of these nodes are now dependent on the cost of the entering arc and thus all nodes which are in the rehung portion of the tree will have new dual values.

In the *dual updating* technique when flow is augmented on the cycle we check to see if any of the arcs in the cycle change their *character*. If so, we do a complete dual update of all nodes above the common node of the cycle. If not, we will update the duals only on the rehung portion of the tree. Note that if no arc changed its character we do not have to compute c′

since it remains the same. We now present our scheme for updating the duals.

<u>Step 9 — **DU**: Updating the duals</u>

```
$e_j \leftarrow$ entering arc
if ($\exists$ an arc $e_k \in C(e_j)$ that has changed its *character*) then
    $c \leftarrow$ *common* node for $C(e_j)$
    n1 $\leftarrow c$
    n2 $\leftarrow$ next(n1)

    while (height(n2) > height($c$)) do:
        $e_k \leftarrow$ arcid(n2)
        $c'_k \leftarrow$ current penalty coefficient for arc $e_k$ (section 1)
        $\pi'_{n2} = \begin{cases} \pi'_{n1} + c'_k & if\ n2 = F(e_k) \\ \pi'_{n1} - c'_k & if\ n2 = T(e_k) \end{cases}$
        n2 $\leftarrow$ next(n2)
        n1 $\leftarrow$ down(n2)
    enddo
else
    p $\leftarrow$ node through which $e_j$ is attached to the tree
    n1 $\leftarrow$ p
    n2 $\leftarrow$ next(n1)
    if ($e_j$ is directed away from the root) then
        $\overline{c'_j} \leftarrow -\overline{c'_j}$
    endif
    while (height(n2) > height(p)) do
        (Note that we are computing the duals only on the rehung portion of the tree)
        $\pi'_{n2} \leftarrow \pi'_{n2} + \overline{c'_j}$
        n2 $\leftarrow$ next(n2)
        n1 $\leftarrow$ down(n2)
    enddo
endif
```

4 Cycle Tracing Technique

In the *cycle tracing technique* the duals maintained are the normal duals. These are based only on the original arc costs and thus do no reflect the infeasibilities that may be present in the tree. The solution process, as in the ***dual updating technique***, can be initiated with any basic solution $\mathbf{x}$ which need not be feasible with respect to the arc bounds. The duals are now determined for $\mathbf{T}_B$ in a manner identical to that in step 3 of **DU**, except that for each arc $e_k \in \mathbf{B}$ we use the original arc cost c_k instead of the penalty coefficient c'_k. Thus step 3 of the network penalty algorithm is the same for both **DU** and **CT** implementations with the above modifications.

Step 4 of the network penalty algorithm selects the entering variable. Since the duals are incorrect, the pricing operation is more complicated than in the **DU** implementation. In the **CT** implementation we price each nonbasic arc using the normal duals. We have two possible outcomes for this operation. If we find any arc that appears to be improving under this pricing scheme, we choose an arc that affords the best improvement and make that arc a candidate to enter. But, we now have to confirm that it is indeed an improving direction by computing its correct reduced cost. We do this by tracing the cycle created by adding this arc to $\mathbf{T}_B$ and determining the penalties that are incurred by the infeasible arcs (if any) present on that cycle. If it is, indeed, improving, we then choose it to be the entering variable.

The second outcome that is possible is that there are no improving arcs using this pricing scheme. This unfortunately does not guarantee that there are no improving directions, in the presence of infeasible arcs in $\mathbf{T}_B$. In that case we may have to consider each nonbasic arc in turn and trace the cycle created by adding it to $\mathbf{T}_B$ in order to determine its correct reduced cost, before ruling it non-improving. This leads to unnecessary cycle traces. We present a simple bound that enables us to eliminate some non-essential cycle traces.

Proposition 4.1:
Let n be the number of infeasible arcs in $\mathbf{T}_B$. Let $\alpha > 0$ be the penalty associated with every unit of infeasible flow. Let $\overline{c_j}$ be the reduced cost of a nonfavorable nonbasic arc e_j. That is, $\overline{c_j} > 0$ if $e_j \in L$ and $\overline{c_j} < 0$ if $e_j \in U$. If also, $\overline{c_j} - (n \times \alpha) > 0$ for $e_j \in \mathbf{L}$ and $\overline{c_j} + (n \times \alpha) < 0$ for $e_j \in \mathbf{U}$, then e_j cannot provide an improving direction. □

Thus any nonbasic arc that satisfies the above criterion can be eliminated from consideration and the corresponding cycle trace can be avoided. However, the above bound does not eliminate all non-essential cycle traces. We now present a selection criteria that can be used to determine the entering variable.

Step 4.1 — CT: Selection criterion for determining a candidate for the entering variable

```
newpr ← 0
n     ← number of infeasible arcs
for (every nonbasic arc $e_j$) do:
    $\overline{c_j} \leftarrow c_j - \pi_{F(e_j)} + \pi_{T(e_j)}$
    if($e_j \in U$) then
      $\overline{c_j} \leftarrow -\overline{c_j}$
    endif
    if ($\overline{c_j} + (n \times \alpha) > newpr$) then
      enter ← $e_j$
      newpr ← $\overline{c_j} + (n \times \alpha)$
    endif
enddo
if (newpr $\neq$ 0) then
    add $e_j$ to $\mathbf{T}_B$
    determine $\overline{c_j'}$ by doing a cycle trace on $C(e_j)$
else
    (stopping criterion: Step 10 of the penalty algorithm)
    if (n > 0) increase $\alpha$ and restart the algorithm (return to step 1)
    if (n = 0) terminate with an optimal solution
endif
```

In our implementation, instead of pricing all the nonbasic arcs and picking the best among them, as indicated above, we price a set of nonbasic arcs and pick the best in that set. Arcs having the same *to* node constitutes one set. Having selected a candidate arc to enter the basis, we now need to determine if that arc indeed provides us with an improving direction. We do this by considering the cycle created by adding this arc to the basis and determining the contribution of any infeasible arcs in the cycle to the reduced cost of the potential candidate . After accounting for this contribution, if the arc still provides an improving direction we will choose it to enter, else we return to the selection criterion and consider the next nonbasic arc. If no nonbasic arc can provide an improving direction then we terminate either with an optimal solution or increase alpha and return to step 1 of the penalty algorithm. We now present an implementation for determining the correct reduced cost $\overline{c'_j}$ for the candidate entering arc e_j by tracing the cycle $C(e_j)$.

Step 4.2 — CT: Trace the cycle $C(e_j)$

```
$e_j$ $\leftarrow$ candidate entering arc
$\overline{c_j}$ $\leftarrow$ reduced cost using normal pricing
$\hat{c}$ $\leftarrow$ 0
$C(e_j) \leftarrow \begin{cases} FC(e_j), & if\ e_j \in \mathbf{L} \\ RC(e_j), & if\ e_j \in \mathbf{U} \end{cases}$
for ( $\forall e_k \in (C(e_j) \cap \mathbf{T}_B)$ ) do:
    if ($e_k$ is a forward arc) then
        $\hat{c} \leftarrow \hat{c} + \alpha \quad if\ x_k > u_k$
        $\hat{c} \leftarrow \hat{c} - \alpha \quad if\ x_k < 0$
    elseif ($e_k$ is a reverse arc) then
        $\hat{c} \leftarrow \hat{c} - \alpha \quad if\ x_k > u_k$
        $\hat{c} \leftarrow \hat{c} + \alpha \quad if\ x_k < 0$
    endif
enddo
if ($e_j \in$ L) then
    $\hat{c} \leftarrow -\hat{c}$
endif
$\overline{c'_j} \leftarrow \overline{c_j} + \hat{c}$
if ($\overline{c'_j}$ is improving) then
    choose $e_j$ to enter
else
    discard $e_j$
endif
```

As in the *dual updating* technique, following the flow augmentation, the duals have to be updated. In this case the updating has to be done only for the rehung portion of the tree and the implementation is similar to the situation in the ***dual updating technique*** when no arc in the cycle has changed its ***character***.

5 Generation of Test Problems:

In generating the test problems for evaluating the performance of the network penalty algorithm, we had a specific goal in mind. We were interested in evaluating the performance of the algorithm mainly in solving capacitated minimum cost network flow problems, although we did test its performance on both assignment and transportation problems.

We used the network generator NETGEN [6] to generate our test problems. We generated three sets of problems. Problem set A consisted of 10 capacitated minimum cost network flow problems, with 400 nodes and 3000 arcs. The problems had 95% of the arcs capacitated. Problems 1-5 had a cost range of 1-10000, while problems 6-10 had costs ranging from 1-5000. In the above two groups, the problems progressively had relaxed arc capacities. That is, in problems 1-5, problem 1 was the most tightly capacitated while problem 5 had the widest range of arc capacities. Problem 1 had arc capacities varying from 25%-50% of the average supply at the source nodes, while for problem 5 the arc capacities were more relaxed, varying from 150%-200% of the average supply. The arc capacities varied similarly for problems 6-10. Table 1 lists the parameters that are common for the above 10 problems. Table 2 gives the problem parameters that vary in the above problems.

TABLE 1: Problem parameters common to the problems in problem set A.

Cost range = 1-10000 for problems 1-5 and 1-5000 for problems 6-10

No. of Nodes	No. of Sources	No. of Sinks	No. of Arcs	Total Supply	Tran shipments		% High Cost Arcs	% Arcs Capac.
					Sources	Sinks		
400	20	80	3000	400000	5	20	10	95

TABLE 2: Variable problem parameters for problems in set A

Prob. No.	Random # Seed	Upper Bound Range	
		Min.	Max.
A1	13502460	5000	10000
A2	27895462	10000	15000
A3	48739284	15000	20000
A4	74837287	20000	30000
A5	38293723	30000	40000
A6	38293723	5000	10000
A7	27895462	10000	15000
A8	48739284	15000	20000
A9	74837287	20000	30000
A10	38293723	30000	40000

Problem set B consists of 3500 nodes and 15000 arcs capacitated minimum cost network flow problems. There are 5 problems in this set and the philosophy behind the choice of their parameters is similar to that for problem set A. Table 3 gives the common data for the above problems, while table 4 lists the parameters that vary for the above problems. Problem set C consists of the standard NETGEN problems 1-35. These are the same problems as described in [6]. The parameters for problem set C are not included here and can be obtained from [6]. However since we use the random number generator on our system, the optimal objective values obtained are different from those given in [6]. These values are given later when we discuss the results.

TABLE 3: Problem parameters common to the problems in problem set B

No. of Nodes	No. of Arcs	Cost Range		Total Supply	% High Cost Arc	% Arcs Capac.
		Min.	Max.			
3500	15000	1	5000	1400000	10	95

TABLE 4: Variable problem parameters for problems in set B

Prob. No.	Random No. Seed	No. of Sources	No. of Sinks	Transshipments		Upper Bound Range	
				Sources	Sinks	Min.	Max.
B1	84635377	70	280	18	70	5000	10000
B2	54575469	100	300	20	75	10000	15000
B3	75635879	60	200	10	45	15000	20000
B4	45657670	100	200	10	45	20000	30000
B5	65855524	80	170	10	45	30000	40000

6 Computational Results:

We compared the performance of both the *dual updating* and the *cycle tracing* implementations of the penalty algorithm with that of the network simplex. We used the network simplex code **NETFLO** of Kennington and Helgason [1] in our testing. Both the penalty implementations were built with **NETFLO** as the base, thereby ensuring that all three codes shared very similar data structures and starting procedures. In the penalty implementations we set the penalty α to be β times the input parameter **maximum arc cost**, where β is usually in the range 0.5 - 3.0. In trying to determine a suitable β that would minimize the total number of iterations required, each problem was solved over the above range and for each case we report the ones that yielded the best results.

The three codes were first tested on the problems in set A. The ***dual updating*** method was able to produce an improvement ranging from 8% - 15% in terms of the number of iterations needed to solve to optimality, over **NETFLO**. The results for the **DU** implementation on problems in set A are reported in table 5. **DU** out performed **NETFLO** in terms of the number of iterations required, and excluding problems 6A and 7A it produced improvements ranging from 8% - 19%. It was able to produce only a 5% improvement in the case of problem 6A and 0.84% improvement in the case of 7A. **NETFLO** however was faster on all of the problems, **DU** taking about 25% - 50% more time. This is due to the additional updating operations that are involved in the penalty codes. This makes each iteration of the penalty code more expensive. In the dual updating technique there is an additional overhead in step 9, where the duals are updated. The **DU** implementation resorts to a more extensive dual update whenever an arc on the augmenting cycle changes its "character" and this happens quite frequently. This additional work load is more apparent in problems 3A and 6A where we find that the β that results in the smallest number of iterations does not also produce the fastest solution time. This is because the fewer iteration solution takes more dual updates than the one that takes more iterations but is faster. We will present a more detailed analysis towards the end of this section.

TABLE 5: Performance of **DU** vs. **NETFLO** on problems in set A

Prob. No.	NETFLO		DU			% Imp. Iter	T_D/T_N	Objective Value
	Iter.	Time, T_N (Seconds)	β	Iter.	Time, T_D (Seconds)			
1A	2147	2.24	1.95	1846	2.87	14.02%	1.28	2268269087
2A	2047	2.01	2.00	1648	2.64	19.49%	1.31	2397270438
3A	1815	1.62	1.27	1527	2.35	15.87%	1.45	2117538641
			1.80	1609	2.28	11.35%	1.41	
4A	1430	1.29	0.90	1224	1.89	14.41%	1.46	1936741441
5A	1344	1.20	1.35	1233	1.82	8.26%	1.52	1846214623
6A	2023	2.03	1.23	1922	3.10	4.99%	1.53	1261636794
			1.90	2021	2.96	0.10%	1.46	
7A	1665	1.62	2.10	1651	2.50	0.84%	1.54	1191949354
8A	1656	1.46	1.80	1501	2.09	9.36%	1.43	911586498
9A	1484	1.41	1.11	1351	2.00	8.96%	1.42	992651321
10A	1430	1.27	2.10	1213	1.81	15.17%	1.42	11128832934

The performance of the *cycle tracing* implementation **CT** on the problems in set A are given in table 6. The improvements in terms of the number of iterations required with respect to that of **NETFLO** is smaller than that obtained with **DU**. The improvements ranged from 5% - 14% and in the case of problems 6A and 9A it was -12.6% and 0.0% respectively. The reason for the poor performance of **CT** is not very apparent. We expected its performance to be on par with that of **DU**. Time wise also, its performance was inferior to that of **DU**. But this is to be expected because of the inefficient pricing technique that is used. Step 4.2 detailed in section 4 is the source of the large overhead associated with the **CT** implementation. Since the duals are not the right duals, a cycle trace is necessary to determine if an entering arc is indeed an improving one. This is an expensive operation, and not always fruitful. Implementation of proposition 4.1 tries to limit the number of wasted or "null" cycle traces. But it is a very loose bound and though it is able to eliminate a majority of the unnecessary cycle traces it is not completely successful. As in the **DU** implementation, the β that produces the best improvement in the number of interations, does not always correspond to the one that produces the fastest solution times. A major reason for this is the number of null cycle traces that are performed. Again, a more detailed analysis is presented at the end of this section.

TABLE 6: Performance of **CT** vs. **NETFLO** on problems in set A

Prob. No.	NETFLO		CT			% Imp. Iter	T_C/T_N	Objective Value
	Iter.	Time, T_N (Seconds)	β	Iter.	Time, T_C (Seconds)			
1A	2147	2.24	1.26	1962	3.82	8.62%	1.71	2268269087
			1.55	1995	3.52	7.08%	1.57	
2A	2047	2.01	1.57	1932	3.68	5.62%	1.83	2397270438
3A	1815	1.62	2.31	1651	2.80	9.04%	1.78	2117538641
4A	1430	1.29	1.59	1343	2.26	6.08%	1.75	1936741441
			1.63	1316	2.49	7.97%	1.93	
5A	1344	1.20	1.55	1272	2.11	5.36%	1.76	1846214623
			1.95	1288	1.99	4.17%	1.66	
6A	2023	2.03	2.40	2278	3.73	-12.6%	1.84	1261636794
7A	1665	1.62	2.45	1551	2.86	6.85%	1.77	1191949354
8A	1656	1.46	2.20	1521	2.58	8.15%	1.79	911586498
9A	1484	1.41	2.67	1483	2.50	0.00%	1.77	992651321
10A	1430	1.27	1.35	1220	2.09	14.69%	1.65	11128832934

Figure 1 illustrates the improvements achieved by the **DU** and **CT** implementations of the penalty algorithm, over **NETFLO**, in terms of the number of iterations required to solve the problem to optimality. The **DU** implementation performs much better than **CT** on all but one of the problems, and out performs **NETFLO**. Figure 2 plots the % improvement in the number of iterations for each problem, obtained using **DU** and **CT**. Figure 3 plots the ratio of the penalty time to the network simplex time for each problem, using the two implementations. Obviously **DU** is better than **CT**, though it is inferior to the **NETFLO** times.

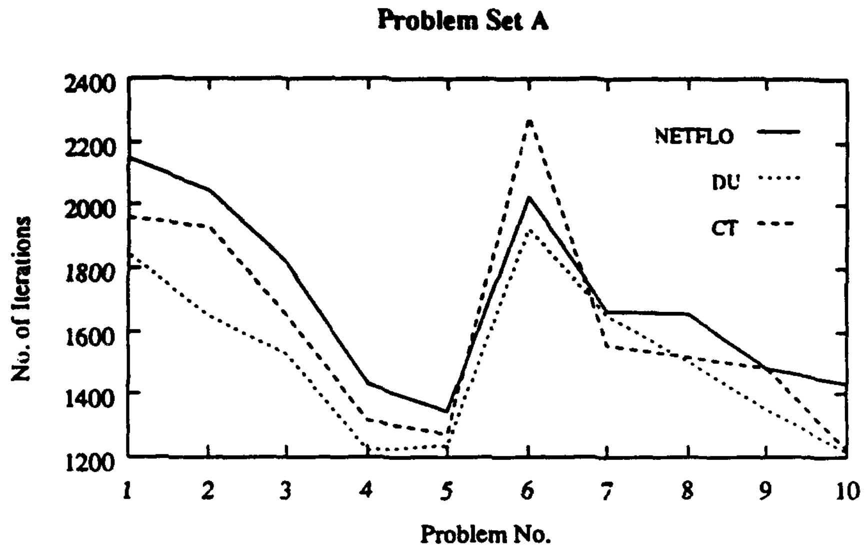


Figure 1: Iterations: Total number of iterations required

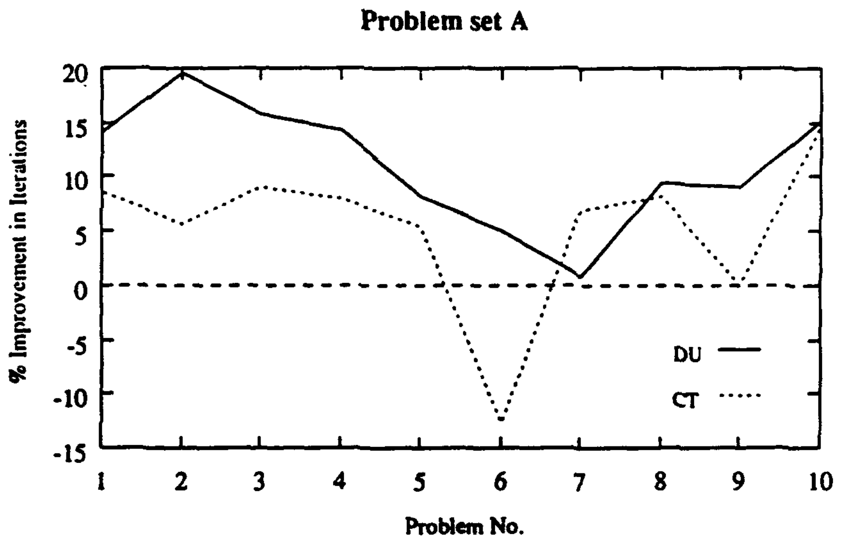


Figure 2: Iterations: % improvement over NETFLO

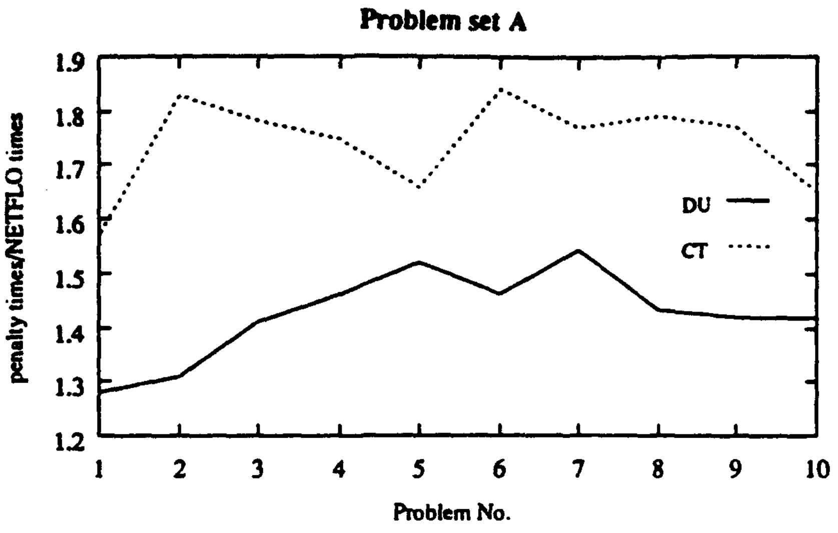


Figure 3: Time: Run time efficiency of DU and CT to NETFLO

We next tested the penalty codes on the standard NETGEN problems [6]. Problems 1-35 of [6] were solved using both **DU** and **CT** and their performance compared with that of **NETFLO**. Problems 1 - 5 are 100 × 100 transportation problems; problems 6 - 10 are 150 × 150 transportation problems; problems 11 - 15 are 200 × 200 assignment problems; problems 16 - 27 are 400 node capacitated network problems; and problems 28 - 35 are uncapacitated 1000 and 1500 node network problems. The maximum arc cost allowed was set to 10000 instead of 100. This is to keep our testing consistent with that in [2]. The results of this testing are given in table 7 for **DU** and table 8 for **CT**. The performance of the two penalty implementations were fairly similar on the above test set, though **DU** had better performance times than **CT** on all but the uncapacitated network problems. **NETFLO** out performed both **DU** and **CT** with regards to running times on all the above problems. Both implementations produced good improvements in the number of iterations required for the assignment problems. The performance on the transportation problems 1 - 10 was not very consistent. For the capacitated network problems 16 - 27 **DU** did better than **CT**. This is because, in capacitated problems there are more infeasibili- ties and the overhead of the **DU** implementation is less than that of the **CT** implementation. Also for the uncapacitated network problems, the **CT** implementation takes less time than the **DU** implementation. This is because, under more simplex-like circumstances the number of "null" cycle traces are much less and the penalty overhead in the **CT** technique is to some extent reduced.

TABLE 7: Performance of DU vs. NETFLO on problems in set C

Prob. No.	NETFLO		DU			% Imp. Iter	T_D/T_N	Objective Value
	Iter.	Time, T_N (Seconds)	β	Iter.	Time, T_D (Seconds)			
1C	672	0.79	0.75	564	0.89	16.08%	1.13	20616842
2C	710	0.82	0.71	703	1.14	1.00%	1.39	19782567
3C	773	0.93	0.75	753	1.34	2.59%	1.44	14680668
4C	691	0.91	0.75	646	1.15	6.51%	1.26	13729316
5C	779	1.10	0.65	724	1.46	7.07%	1.33	11525755
6C	1391	2.08	0.82	1191	2.54	14.38%	1.22	22228918
7C	1386	2.45	1.00	1254	3.03	9.52%	1.24	16272727
8C	1328	2.38	0.78	1283	3.27	3.39%	1.37	18400181
9C	1503	2.95	0.75	1426	3.76	5.12%	1.28	12040215
10C	1496	2.98	0.84	1293	3.61	13.57%	1.21	14695801
11C	2242	1.52	0.61	1729	1.97	22.88%	1.30	47352
12C	2439	1.72	0.95	2049	2.36	16.00%	1.37	326853
13C	2495	1.91	0.72	1941	2.69	22.20%	1.41	248242
14C	2359	2.10	0.65	2086	2.95	11.57%	1.41	247036
15C	2335	2.29	0.72	2074	3.19	11.18%	1.39	212597
16C	963	0.63	1.58	812	1.08	15.68%	1.72	6815524469
17C	1035	0.70	2.40	918	1.06	11.30%	1.53	2646770386
18C	923	0.62	1.40	754	0.94	18.31%	1.53	6663684919
			1.65	807	0.88	12.57%	1.44	
19C	1053	0.71	1.20	840	1.02	20.23%	1.43	2618979806
20C	1338	0.98	1.34	1087	1.70	18.76%	1.74	6749969302
21C	1262	0.96	1.08	971	1.28	23.06%	1.34	2631027973
22C	959	0.72	1.32	810	1.24	15.54%	1.74	6621515104
23C	1312	0.94	1.15	938	1.35	28.51%	1.44	2630071408
24C	1775	0.98	2.02	1465	1.58	17.46%	1.62	6829799687
25C	2507	1.57	1.20	2161	2.76	13.80%	1.76	6396423129
26C	1085	0.54	1.82	996	1.02	8.20%	1.91	5297702923
27C	1900	1.07	1.43	1555	1.79	18.16%	1.69	4863992745
28C	3450	3.21	1.82	3234	5.71	6.26%	1.78	11599233408
29C	3769	3.32	1.18	3278	8.22	13.03%	2.48	11700773092
			2.10	3470	6.14	7.93%	1.85	
30C	4465	3.65	1.36	4002	8.05	10.37%	2.20	8782721260
31C	4354	3.55	1.32	3818	8.08	12.31%	2.28	8577913734
			1.86	4133	6.99	5.08%	1.97	
32C	5704	7.06	1.95	5086	14.10	10.83%	2.00	17996365110
33C	5964	6.93	1.44	5039	15.10	15.51%	2.18	184248939300
34C	7933	8.36	2.05	6418	17.73	16.20%	2.12	14596094907
35C	7692	8.88	1.09	6499	23.03	15.51%	2.59	143359003861
			1.79	6991	16.60	9.11%	1.87	

TABLE 8: Performance of CT vs. NETFLO on problems in set C

Prob. No.	NETFLO		CT			% Imp. Iter	T_C/T_N	Objective Value
	Iter.	Time, T_N (Seconds)	β	Iter.	Time, T_C (Seconds)			
1C	672	0.79	0.82	608	1.19	9.52%	1.51	20616842
2C	710	0.82	0.83	672	1.30	5.35%	1.60	19782567
3C	773	0.93	0.79	746	2.22	3.49%	2.39	14680668
			0.81	772	1.60	0.13%	1.72	
4C	691	0.91	0.90	671	1.52	2.89%	1.69	13729316
5C	779	1.10	0.65	723	1.77	7.19%	1.62	11525755
6C	1391	2.08	0.86	1199	2.92	13.80%	1.40	22228918
7C	1386	2.45	0.92	1211	3.59	12.63%	1.45	16272727
8C	1328	2.38	1.45	1354	4.07	-1.96%	1.70	18400181
9C	1503	2.95	0.82	1502	4.94	0.07%	1.67	12040215
10C	1496	2.98	1.40	1382	4.49	7.62%	1.49	14695801
11C	2242	1.52	0.75	1807	2.51	19.40%	1.63	47352
12C	2439	1.72	0.90	1779	2.64	27.06%	1.56	326853
13C	2495	1.91	0.73	2076	3.19	16.79%	1.69	248242
14C	2359	2.10	0.51	1814	4.76	23.10%	2.23	247036
			0.55	2092	3.79	11.32%	1.78	
15C	2335	2.27	0.85	2334	4.38	0.04%	1.93	212597
16C	963	0.63	1.20	716	1.15	25.65%	1.83	6815524469
			2.35	929	1.10	3.53%	1.75	
17C	1035	0.70	2.60	988	1.31	4.54%	1.87	2646770386
18C	923	0.62	1.38	775	1.04	16.03%	1.68	6663684919
19C	1053	0.71	1.55	925	1.31	12.16%	1.82	2618979806
20C	1338	0.98	2.14	1195	1.78	10.69%	1.80	6749969302
21C	1262	0.96	106	974	1.39	22.82%	1.38	2631027973
22C	959	0.72	1.98	911	1.33	5.01%	1.85	6621515104
23C	1312	0.94	1.26	808	1.38	31.55%	1.45	2630071408
24C	1775	0.98	2.55	1480	1.57	16.62%	1.51	6829799687
25C	2507	1.57	1.70	2250	2.80	10.25%	1.84	6396423129
26C	1085	0.54	2.30	957	0.97	11.80%	1.80	5297702923
27C	1900	1.07	1.22	1483	2.16	21.95%	2.00	4863992745
			2.48	1699	1.94	10.58%	1.80	
28C	3450	3.21	1.32	3082	6.41	10.67%	1.94	11599233408
29C	3769	3.32	1.88	3477	6.67	7.75%	2.02	11700773092
30C	4465	3.65	1.79	4020	7.26	9.97%	1.97	8782721260
31C	4354	3.55	1.84	4190	7.69	3.77%	2.17	8577913734
32C	5704	7.06	2.11	5186	13.36	9.08%	1.82	17996365110
33C	5964	6.93	1.75	5588	21.92	6.30%	3.20	184248939300
			2.16	5772	12.94	3.22%	1.89	
34C	7933	8.36	1.78	7263	17.63	8.45%	2.18	14596094907
35C	7692	8.88	1.83	6932	18.73	9.88%	2.05	143359003861
			2.23	7018	16.08	8.76%	1.76	

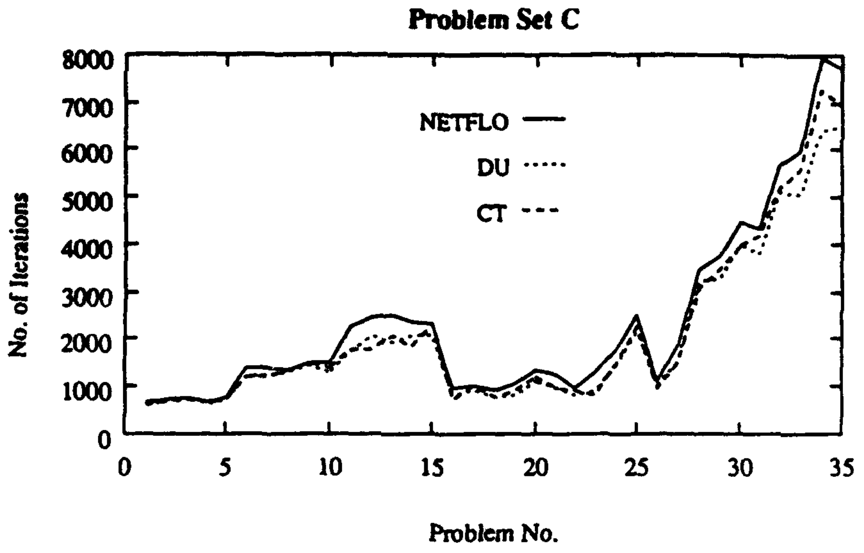


Figure 4: Iterations: Total number of iterations required

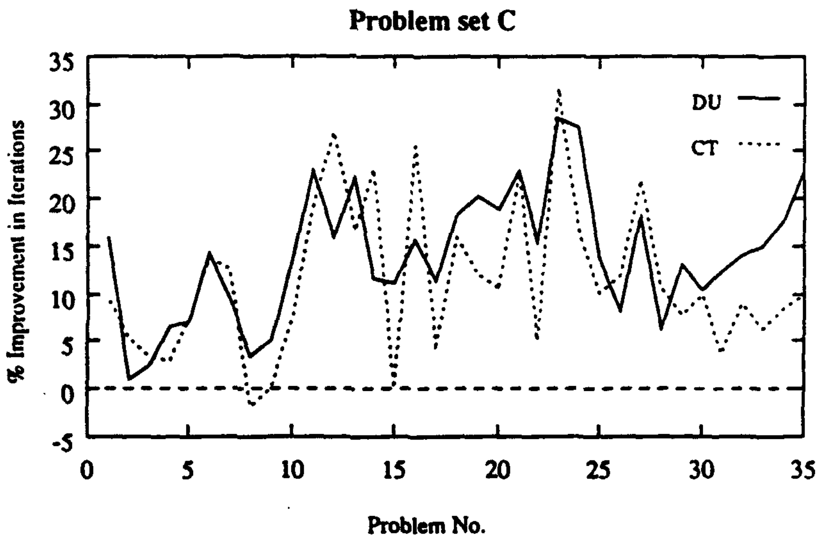


Figure 5: Iterations: % improvement over NETFLO

Figure 4 depicts the performance of **DU** and **CT** with respect to **NETFLO** in terms of the number of iterations required. Figure 5 shows the % improvement, in terms of the number of iterations, achieved by **DU** and **CT** over **NETFLO**, while Figure 6 gives the ratio of the times needed by a penalty implementation to that of **NETFLO** for each problem.

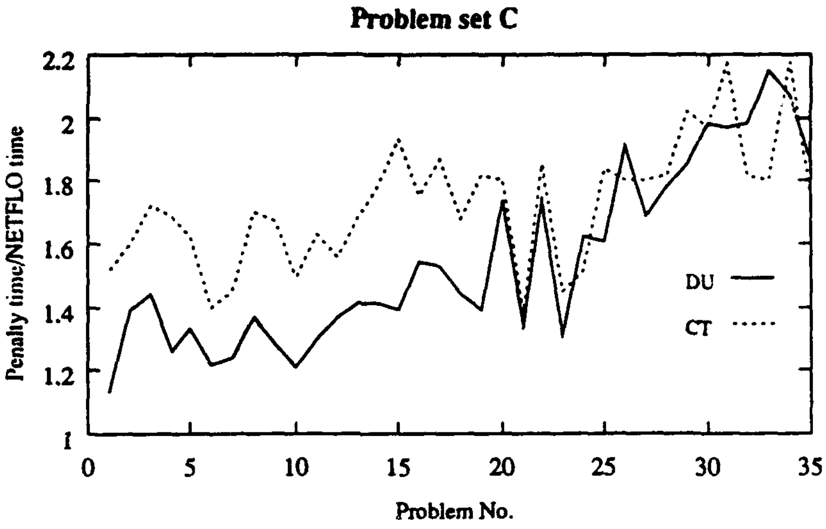


Figure 6: Time: Run time efficiency of **DU** and **CT** to **NETFLO**

At this point it is important to point out a few things. The convergence of the penalty method is very sensitive to the value chosen for the penalty α. Also, in [2] the authors sugggested that the range of the arc costs should influence our choice of α. To this end we decided to tie in the penalty with the maximum arc cost. Also, in our networks, any artificial arcs will have a cost equal to number of nodes in the tree times the maximum arc cost. Thus for trees with a large number of nodes, artificial costs could become very large. Thus the penalty α chosen would be ineffective unless it was able to match these large costs. However too large a value for α inhibits infeasible arc flows and hence defeats the purpose. So we opted to use a two-phase approach while solving the problems in set B. We used the network simplex until we had reduced the flows on all artificial arcs to zero. We then applied the penalty code from that point on. All comparative statistics with **NETFLO** are based starting from this point. Only **DU** was used in testing on problem set B. The results of the testing are summarized in table 9.

TABLE 9: Performance of **DU** vs. **NETFLO** on problems in set B

Prob. No.	NETFLO		DU			% Imp. Iter	T_D/T_N	Objective Value
	Iter.	Time, T_N (Seconds)	β	Iter.	Time, T_D (Seconds)			
1B	26367	69.54	2.31	24143	105.64	8.43%	1.52	8362010359
2B	22328	50.81	2.31	19610	72.52	12.17%	1.43	6721794329
3B	21312	35.94	1.72	19680	70.98	7.66%	1.98	7879914744
			2.05	20221	62.56	5.12%	1.74	
4B	23670	42.35	2.07	20525	78.27	13.29%	1.85	7216585110
			3.00	21034	64.68	11.14%	1.53	
5B	22525	34.02	1.41	19281	81.85	14.40%	2.41	7412956255
			2.89	20364	54.66	9.59%	1.61	

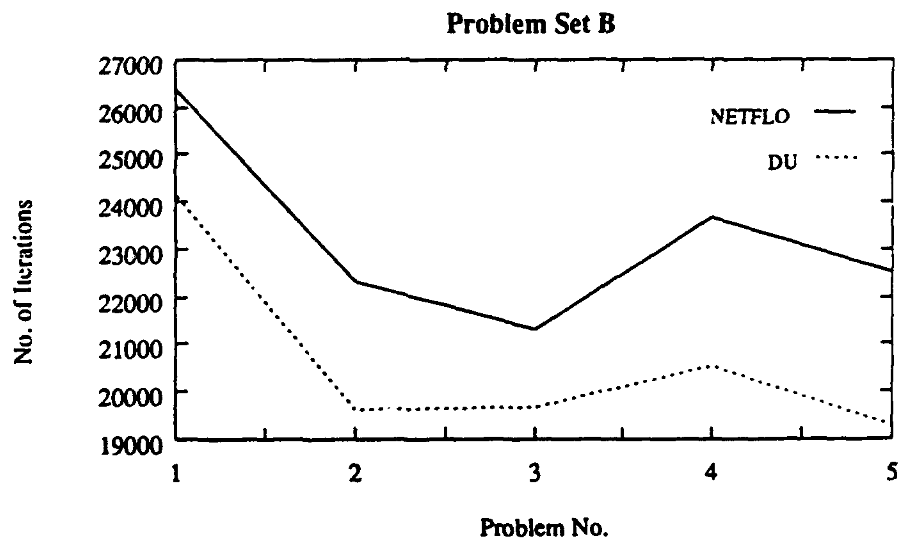


Figure 7: Iterations: Total number of iterations required

Figures 7 and 8 graph the performance of **DU** in solving problem set B.

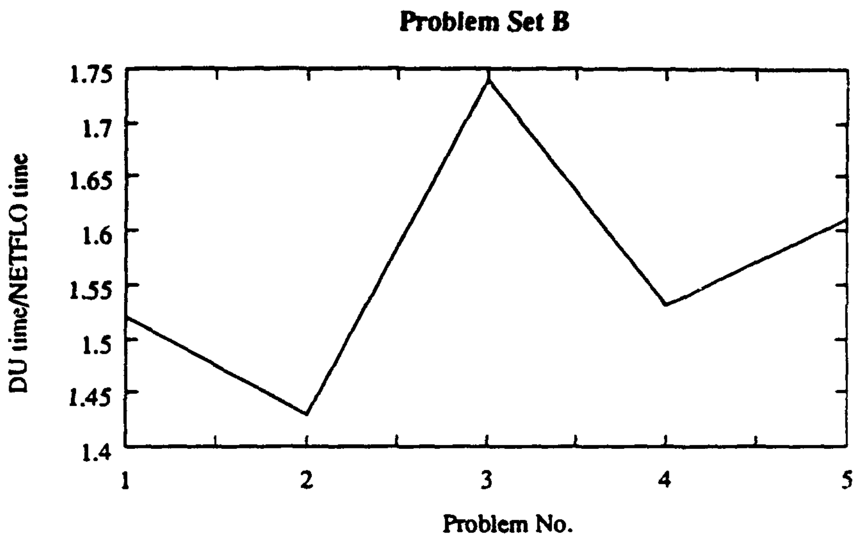


Figure 8: Time: Run time efficiency of **DU** to **NETFLO**

Based on our testing we conclude the following.

- The convergence of the penalty algorithm is very sensitive to the value of the penalty α.

- For large networks, where the initial starting basis has too many artificial arcs, the penalty α has to be high enough to match the artificial costs. This renders the method inefficient. A composite starting feature becomes necessary to overcome this problem.

- The **DU** implementation was more efficient and consistent than the **CT** implementation. Both implementations performed better on problems with a wider range of arc capacities. This is because, in tightly capacitated problems it takes more iterations to remove infeasibilities and the number of infeasibilities that occur are also quite high.

- Though the penalty algorithm does require fewer iterations than the network simplex, the network simplex is more efficient and takes less work per iteration, and this results in faster solution times.

TABLE 10:

Prob. No.	β_L	Dual Updates		β_H	Dual Updates	
		Total	per Iter.		Total	per Iter.
3A	1.80	66206	41	1.27	73131	47
6A	1.90	95980	47	1.23	105724	55
3B	2.05	2579586	127	1.72	2871147	153
4B	3.00	2725966	129	2.07	3081665	150
5B	2.89	2217494	108	1.41	2803285	145
29C	2.10	201850	58	1.18	277826	84
35C	1.79	553704	79	1.09	728892	112

We would now like to present a brief analysis of the work per iteration distribution in the penalty implementations. In tables 5 - 9 we can see that for a few of the problems, the β or the value of the penalty α that produces the best results in terms of the number of iterations does not necessarily correspond to the fastest time solutions. Table 10 above lists some of the problems which exhibited this characterstic with the **DU** implementation. After a careful scrutiny of the various computations that occur during each iteration we determined that the number of duals that have to be updated over all is a good measure of the work done. For each dual update also requires certain other updates on the tree. We collected statistics corresponding to the total number of duals that are updated in a run for both cases, i.e., for the run with β such that the quickest solution is obtained (β_L) and one with the β that converges with fewer iterations but taking more time (β_H). As we can see the quicker solution requires fewer dual updates and hence fewer of the other updating operations. Also, each time the higher value of β produces the faster solution times. This can be explained as follows. Higher β values result in fewer infeasibilities. Thus many flow augmentations are in the feasible realm and this roughly mimics the network simplex. Higher infeasibilities result at lower β values, and while this reduces the number of iterations, the penalty overhead increases the solution time. Figure 9 depicts the average number of dual updates that are carried out per iteration, for the two different β values, for the problems in Table 10.

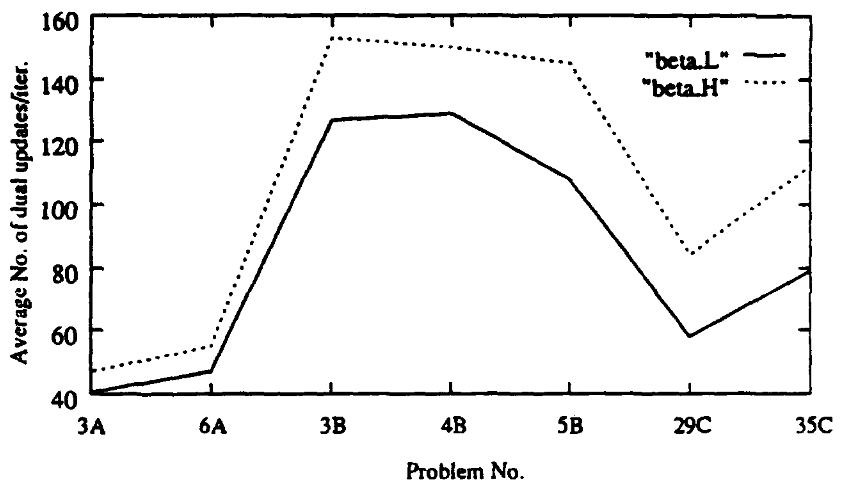


Figure 9: **DU** *implementation:* Dual updates per iteration.

The above anomaly occurs for the **CT** implementation also, but for different reasons. The dual updating in the **CT** implementation is very similar to that in **NETFLO**. So the main overhead is due to the failure to detect "null" cycle traces during the pricing operation. Table 11 compares the number of null cycle traces that are performed for the two values of β. As before β_L corresponds to the run which takes less time and β_H corresponds to the one that takes fewer iterations but more run time.

From the table it is clear that whenever more "null" cycles are traced, the running time also increases. The "saved" column corresponds to the number of null cycles that are prevented by invoking proposition 4.1. Thus, in spite of the rather loose bound it implies, the proposition is quite effective in eliminating unwanted cycle traces. Also, one would expect the bound to function more effectively at lower values of β, since it is tighter. The fact that this does not appear to be the case always is because of the distribution of the infeasible arcs. Figure 10 displays the null cycles that are traced for each β for problems in Table 11.

TABLE 11

Prob. No.	β_L	"Null" Cycles Traced	"Null" Cycles Saved	β_H	"Null" Cycles Traced	"Null" Cycles Saved
1A	1.55	84	33063	1.26	839	33063
4A	1.59	428	27963	1.63	432	31491
5A	1.95	3	27635	1.55	145	28618
3C	0.81	0	30399	0.79	1000	41860
14C	0.55	148	78704	0.51	2538	74632
16C	2.35	20	7794	1.20	664	8747
27C	2.48	64	24146	1.22	907	22925
33C	2.16	2192	39719	1.75	16337	32545
35C	2.23	2660	64501	1.83	5902	51409

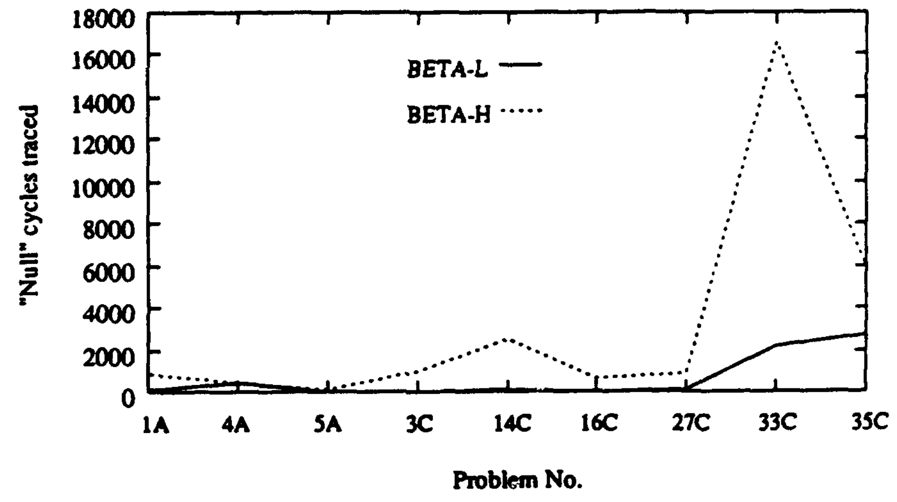


Figure 10: CT implementation: Total number of "null" cycles traced.

It is very clear from the figure that the longer timed runs trace a larger number of "null" cycles as compared to the runs taking less time.

In conclusion we feel that while the network penalty algorithm does provide some improvement in the number of iterations it is still not as efficient as the network simplex. This is due to the associated logic that goes into maintaining the duals and other connected information on the tree, being built into the heart of the optimization routine. Thus each penalty iteration turns out to be more computationally intensive than the corresponding network simplex iteration. We do not claim that ours is the most efficient implementation, but the coding has been uniform all around and therefore we believe our comparisons are fair.

References

[1] Kennington, J.L., and R.V. Helgason, *Algorithms for Network Programming*, John Wiley, New York, 1983.

[2] Gamble, A.B., A.R. Conn, and W.R. Pulleyblank, "A Network Penalty Method", *Math Programming,* North-Holland, 50, 53-73, 1991.

[3] A.R. Conn, "Constrained optimization using a nondifferentiable penalty function", *SIAM Journal of Numerical Analysis,* 10, 760-784, 1973.

[4] A.R. Conn, "Linear programming via a nondifferentiable penalty function," *SIAM Journal of Numerical Analysis,* 13, 145-154, 1976.

[5] R.H. Bartels, " A penalty linear programming method using reduced-gradient basis-exchange techniques", *Linear Algebra and its Applications,* 29, 17-32, 1980.

[6] Klingman, D., A. Napier, and J. Stutz, "NETGEN: A program for generating large-scale capacitated assignment, transportation, and minimum cost network flow problems," *Management Science,* 814-821, 1974.

A Nearly Asynchronous Parallel LP-based Algorithm for the Convex Hull Problem in Multidimensional Space*

José H. Dulá, Richard V. Helgason, and Nandagopal Venugopal

Computer Science and Engineering Department
Southern Methodist University
Dallas, Texas 75275

30 June 1993

Abstract

The convex hull of a set $\mathcal{A}$ of n points in $\Re^m$ generates a polytope $\mathcal{P}$. The frame $\mathcal{F}$ of $\mathcal{A}$ is the set of extreme points of $\mathcal{P}$. The *frame problem*, the identification of $\mathcal{F}$ given $\mathcal{A}$, is central to problems in operations research and computer science. In OR it occurs in specialized areas of optimization theory: stochastic programming and redundancy in linear programming. In CS it is an important problem in computational geometry. The problem also appears in economics and statistics. The frame problem is computationally intensive and this limits its applications. The standard LP-based approaches for identifying $\mathcal{F}$ solve several linear programs with m rows and $n-1$ columns, one for each element of $\mathcal{A}$. In this paper we report on a parallel procedure for identifying $\mathcal{F}$ using a new LP-based approach. The new approach also uses linear programs with m rows, but the

*Acknowledgment. This research was partially supported by the Navy Personnel Research and Development Center and the Office of Naval Research under Contract Number N00014-91-J-1234

linear programs which must be solved begin with a small number of columns and grow in size, never exceeding the number of points of $\mathcal{F}$. On a small set of test problems, the serial time to identify $\mathcal{F}$ varied from one-half to two-thirds that of an enhanced implementation of the standard approach. On those same problems, our parallel MIMD nearly asynchronous implementation achieved a speedup factor of 7 to 8 using 14 to 16 processors. These developments will permit the solution of problems previously considered too large.

Key Words: Convex hull problem, frame, linear programming, data envelopment analysis, asynchronous parallel algorithms, parallel linear programming.

1 Introduction

A given collection of n points $\mathcal{A} = \{a^1, \ldots, a^n\}$ in $\Re^m$ defines or generates a polytope $\mathcal{P}$ of dimension at most m, which is the set of all convex combinations of points of $\mathcal{A}$, also known as the *convex hull* of $\mathcal{A}$, denoted by *con*$\mathcal{A}$. The extreme points of $\mathcal{P}$, a subset of $\mathcal{A}$ which we call the *frame* of $\mathcal{A}$ and denote by $\mathcal{F}$, provides a minimal description of the polytope. We call the identification of $\mathcal{F}$ given $\mathcal{A}$ the *frame problem.* The frame problem appears in equivalent forms in several applications. In operations research the problem appears directly in two important areas of optimization: redundancy in linear programming and stochastic programming. The frame problem is also involved in the econometric methodology for measuring the comparative efficiency among many economic firms known as "data envelopment analysis" (DEA). In computer science the frame problem plays a role in one of the classical problems in computational geometry, that of finding the hyperplanes which define the facets of the convex hull of a finite set of points. Finally, the frame problem appears in statistics in the evaluation of Gastwirth estimators. The role of the frame problem in these applications is presented in more detail in [3].

2 Previous LP-based Approaches

Perhaps the first work to address directly the frame problem in its general form was presented by Wets and Witzgall [8] in the context of the equivalent problem of identifying the generating elements of a convex polyhedral cone. The approach taken by Wets and Witzgall to find the "frame" of the cone is essentially based on simplex method iterations. A more formal algorithm presented in Wallace and Wets [7] is also based on the solution of linear programs.

A more recent work by Rosen, Xue, and Phillips [5] also proposes an algorithm for identifying the extreme points of the convex hull based entirely on linear programs and, in addition, reports numerical results using a parallelization scheme, apparently the first attempt at implementing an LP-based approach to the frame problem in parallel.

Most previous LP-based approaches to the frame problem have essentially relied on the following linear program to determine if element $a^k \neq 0$ of the set $\mathcal{A} = \{a^1, \dots, a^n\}$ is an element of $\mathcal{F}$:

$$\min z_1 = \sum_{\substack{j=1 \\ j\neq k}}^{n} \lambda_j \quad \text{s.t.} \quad \sum_{\substack{j=1 \\ j\neq k}}^{n} a^j \lambda_j = a^k; \quad \lambda_j \geq 0; \; j = 1, \dots, n \qquad \textbf{(LP1)}$$

The following result relates the solution of **LP1** to the determination of the status of $a^k \neq 0$ (for a proof see [3]).

Result 1. *For* **LP1** *feasible, the point $a^k \neq 0$ is an element of the frame $\mathcal{F}$ if and only if the optimal objective function value of* **LP1**, *z_1^*, is greater than* 1.

The linear program formulation **LP1** is a generic form which can be used to resolve whether or not the point $a^k \in \mathcal{A}$ belongs to $\mathcal{F}$. Note that it is possible to identify conclusively the status of all the points in the set $\mathcal{A}$ by solving this linear program n times over all right-hand side vectors $a^1, \dots, a^n$.

Linear programming formulations for solving the frame problem previously proposed are equivalent to **LP1** and the approach utilizing repeated solutions of linear programs such as the one here is standard. For example, in Rosen, Xue, and Phillips [5] the approach is to add the constraint

$\sum_{\substack{j=1 \\ j\neq k}}^{n} \lambda_j = 1$ to formulation **LP1**, discard the objective function and then apply Phase 1 to verify if the set of $m+1$ equalities has a nonnegative solution. The approach presented in Wallace and Wets [7] is also based on verifying feasibility, but since their formulation is for finding the extreme rays of the positive cone generated by the elements of $\mathcal{A}$, the constraint $\sum_{\substack{j=1 \\ j\neq k}}^{n} \lambda_j = 1$ is not needed. The linear programming formulation applied in DEA introduces extra variables, one to measure "efficiency" and the rest used as slacks.

3 Theoretical Aspects of LP-based Approaches

We now summarize some recent results which apply to previous LP-based approaches to the frame problem and to the newer approach we proposed in [3].

We assume that the number of points n is greater than the dimension m with at least one subset of m vectors being linearly independent, and that the convex hull $\mathcal{P}$ contains the origin in its interior (if not, the points can be translated to satisfy this condition). These assumptions are necessary to establish that the polytope $\mathcal{P}$ has dimension m.

Consider the following linear program:

$$\min z_2 = \sum_{j=1}^{n} \lambda_j \quad \text{s.t.} \quad \sum_{j=1}^{n} a^j \lambda_j = b; \quad \lambda_j \geq 0; \; j = 1, \ldots, n \qquad \textbf{(LP2)}$$

where b is an arbitrary nonzero vector in $\Re^m$ and not necessarily one of the elements of $\mathcal{A}$. Notice also that the index j is defined over all its possible values without excluding any as in the original expression for **LP1**. Finally, observe that **LP2** is always feasible and its solution bounded since, by assumption, $con\mathcal{A}$ has full dimension and contains the origin in its interior. Denote by z_2^* the optimal objective function value of **LP2**.

The following two results are proved in [3]:

Result 2. *If z_2^* is the optimal solution to* **LP2** *for some $b \neq 0$ then*

(1) $z_2^* < 1$ *if and only if b is interior to $\mathcal{P}$.*
(2) $z_2^* = 1$ *if and only if b is on the boundary of $\mathcal{P}$.*

(3) $z_2^* > 1$ *if and only if* b *is exterior to* $\mathcal{P}$.

Result 3. *The optimal basis to* **LP2**, *if unique, is composed of points* $a^{j_1}, \ldots, a^{j_m}$ *which are elements of the frame* $\mathcal{F}$.

These results can be directly applied to the formulation **LP1** with two important implications. The first is that any time the original linear ~~gram **LP1** is solved and a unique optimal basis is obtained, the m points of . which are in the basis are revealed as elements of the frame. The second is that every time a point is discovered not to be an element of the frame it can be removed from the linear program formulation. These implications can be used to enhance the performance of the procedure for identifying the frame of $\mathcal{A}$ by reducing the total number of linear programs that need to be solved as well as by reducing their size by removing columns from the matrix of coefficients.

Using the formulation **LP1** and the results accompanying it to enhance it means that it is required that both the objective function value and the basic feasible solution be known to determine whether a point belongs to the frame. The fact that, eventually, an accurate optimal basic feasible solution to **LP1** is required is one reason why interior point methods are not used. Another reason is that the input-output matrix in **LP1** is dense with many more columns than rows. This is a particularly unattractive structure for interior point methods since these are very sensitive to the number of columns.

4 A General Approach

We now assume that some of the elements of the frame are known. (Initial elements could easily be identified by applying simple preprocessing schemes to the set $\mathcal{A}$ as in [4].) With such knowledge, the set $\mathcal{A}$ can be partitioned into three subsets, $\mathcal{A}^E$, $\mathcal{A}^U$, and $\mathcal{A}^N$ where:

$\mathcal{A}^E$ = *the set of all currently known elements of* $\mathcal{F}$,
$\mathcal{A}^N$ = *the set of all currently known nonextreme points of* $\mathcal{P}$,
$\mathcal{A}^U$ = *the set of all other points of* $\mathcal{A}$, *whose status is yet to be assigned.*

Based on this partitioning we define

$\mathcal{P}^E =$ *the convex hull of the points in* $\mathcal{A}^E$, *itself a polytope such that* $\mathcal{P}^E \subset \mathcal{P}$.

Consider the following procedure:

Begin Procedure **ProcessPoint**

Step 1. Select a point $a^k \in \mathcal{A}^U$.

Step 2. Determine if a^k belongs to $\mathcal{P}^E$ (the "current" convex hull). If so, remove (the *interior* point) a^k from $\mathcal{A}^U$, add a^k to $\mathcal{A}^N$, and exit the procedure.

Step 3. Generate a direction $v \in \Re^m$ that is normal to a hyperplane separating a^k and $\mathcal{P}^E$.

Step 4. Calculate the maximum of the inner products $\langle v, a^p \rangle; \forall a^p \in \mathcal{A}^U$. Let $\mathcal{A}^{max}$ be the set of all points of $\mathcal{A}^U$ which attain this maximum. Identify one or more extreme points of the set $\mathcal{A}^{max}$ itself. Remove all such identified points from $\mathcal{A}^U$ and add them to $\mathcal{A}^E$.

Step 5. Calculate the minimum of the inner products $\langle v, a^p \rangle; \forall a^p \in \mathcal{A}^U \cup \mathcal{A}^E$. Let $\mathcal{A}^{min}$ be the set of all points of $\mathcal{A}^U \cup \mathcal{A}^E$ which attain this minimum. Identify one or more extreme points of the set $\mathcal{A}^{min}$ itself. Remove all such identified points which are also from $\mathcal{A}^U$ and add them to $\mathcal{A}^E$.

End Procedure

The following results justify and show how Steps 4 and 5 of procedure **ProcessPoint** may be implemented:

Result 4. *If the maximum in Step 4 of Procedure* **ProcessPoint** *occurs at a unique point, a new element of* $\mathcal{F}$ *is generated.*

Result 5. *If the minimum in Step 5 of Procedure* **ProcessPoint** *occurs at a unique point and that point is from* $\mathcal{A}^U$, *a new element of* $\mathcal{F}$ *is generated.*

The above results follow from the fact that optimal solutions to the linear programs

$$\max_{x \in P \subset \Re^m} \langle c, x \rangle, \quad \text{and} \quad \min_{x \in P \subset \Re^m} \langle c, x \rangle,$$

for some $c \neq \hat{0}$, an arbitrary vector in $\Re^m$, and P any nonempty polytope, must occur at extreme points of P. When the maximum or minimum is unique, the point where this value is attained is necessarily an extreme point of the polytope P. In the case of the maximum, if $H(v, \beta)$ is a supporting hyperplane separating the exterior point, a^k, from the current polytope, such that the polytope belongs to the associated open halfspace $H^{--}(v, \beta) = \{y \in \Re^m | \langle v, y \rangle < \beta\}$, then since $\langle v, a^k \rangle > \beta$, either $\langle v, a^k \rangle$ is the maximum value for the inner product or the maximum is attained at some other point of $\mathcal{A}^U$. In any case, the maximum is attained and, if unique, it is necessarily an extreme point of $\mathcal{P}$ and an element of the frame.

The possibility of ties among eligible points in the maximum or minimum value of the inner products in Steps 4 and 5 presents a complication. If there is a tie among several points from the reference hyperplane, it may not be immediately possible to identify which of the points participating in the tie are extreme points of P. The following result shows how this can be resolved in essentially a recursive manner (for a proof see [4]).

Result 6. *Suppose that exactly T points, $\tilde{a}^1, \ldots, \tilde{a}^T$, participate in a tie for the farthest distance (on the same side) from a reference hyperplane H in Step 4 or 5 of procedure* **ProcessPoint**. *Then $\tilde{a}^j$ is an extreme point of $\mathcal{P}$ if and only if $\tilde{a}^j$ is an extreme point of $U = \text{con}\{\tilde{a}^1, \ldots, \tilde{a}^T\}$.*

This result indicates that the resolution of ties reduces to a smaller version of our original frame problem. The resolution of ties is an implementation problem. Note that if only two points are involved in a tie they are both necessarily extreme points of $\mathcal{P}$.

A general algorithm for identifying $\mathcal{F}$ is now apparent. The procedure **ProcessPoint** is simply repeated until $\mathcal{A}^U$ becomes empty. This algorithm must solve the frame problem since at least one point leaves $\mathcal{A}^U$ in either Step 2 or Step 4. Such an algorithm with a declared objective to *not use* linear programs was implemented and computational results were reported in [4].

5 The New LP-based Approach

We recently [3] proposed a new procedure for solving the frame problem based on the solutions to linear programs for the case of a polytope of full dimension. On a small set of test problems, the (serial) time to identify $\mathcal{F}$ varied from one-half to two-thirds that of an enhanced implementation of the standard approach.

Our new LP-based approach to the frame problem relies on the following linear program:

$$\min z_3 = \sum_{j=1}^{\hat{n}} \lambda_j \quad \text{s.t.} \quad \sum_{j=1}^{\hat{n}} \hat{a}^j \lambda_j = a^k; \quad \lambda_j \geq 0; \; j = 1, \ldots, \hat{n} \qquad \textbf{(LP3)}$$

where $\hat{a}^1, \ldots, \hat{a}^{\hat{n}}$ are the elements of $\mathcal{A}^E$, $\hat{n} \geq m+1$, $\mathcal{P}^E$ has dimension m and contains the origin, and $a^k \in \mathcal{A}^U$.

The following result shows how Step 3 of procedure **ProcessPoint** may be implemented following the use of **LP3** for Step 2 (a proof is given in [3]):

Result 7. *An optimal, dual-feasible, basis for* **LP2** *for an exterior point a^k defines a supporting hyperplane for $\mathcal{P}^E$ that separates it from a^k. Moreover, this hyperplane is given by $H(\hat{\pi}^*, 1)$ where $\hat{\pi}^*$ is the corresponding optimal dual solution.*

We may now state the new LP-based procedure:

Begin Procedure **LPFindFrame**

```
Step 0. If $\mathcal{A}^U$ is empty, exit the procedure.
Step 1. Select a point $a^k \in \mathcal{A}^U$.
Step 2. Determine if $a^k$ belongs to $\mathcal{P}^E$ by solving LP3. If so, remove $a^k$ from
        $\mathcal{A}^U$, add $a^k$ to $\mathcal{A}^N$, and return to Step 0.
Step 3. Generate the direction $v \in \Re^m$ normal to a hyperplane separating $a^k$
        and $\mathcal{P}^E$, by setting $v = \pi^*$, the optimal dual solution to LP3.
Step 4. Calculate the maximum of the inner products $\langle v, a^p \rangle; \forall a^p \in \mathcal{A}^U$. Let
        $\mathcal{A}^{max}$ be the set of all points of $\mathcal{A}^U$ which attain the maximum. Iden-
        tify one or more extreme points of the set $\mathcal{A}^{max}$ itself. Remove all
        such identified points from $\mathcal{A}^U$ and add them to $\mathcal{A}^E$.
```

Step 5. Calculate the minimum of the inner products $\langle v, a^p \rangle; \forall a^p \in \mathcal{A}^U \cup \mathcal{A}^E$. Let $\mathcal{A}^{min}$ be the set of all points of $\mathcal{A}^U \cup \mathcal{A}^E$ which attain the minimum. Identify one or more extreme points of the set $\mathcal{A}^{min}$ itself. Remove all such identified points which are also from $\mathcal{A}^U$ and add them to $\mathcal{A}^E$.

Step 6. If $a^k \in \mathcal{A}^U$, return to Step 2. Otherwise return to Step 0.

End Procedure

Note that in Step 4, point a^k may or may not have been identified as an extreme point and its status is checked in Step 6. Further, if the status of a^k remains undetermined, we have chosen to continue with that choice of a point from $\mathcal{A}^U$ by returning to Step 2, rather than Step 1. This choice allows us to make use of the advanced feasible basis still available from the last LP solution in Step 2, since we have only added one or more columns to the previous LP.

The status of point a^k will eventually be resolved since as the procedure cycles between Steps 2 and 6, at least one point leaves $\mathcal{A}^U$ in either Step 2 or Step 4. The same argument shows that this procedure must eventually solve the frame problem.

The difference between the standard LP-based approach and this new procedure is that, in the standard approach, every iteration works with the "whole" polytope, extracting at least one point at each iteration, and applies a check to determine if such a point is extreme or not. In the new procedure proposed here, the polyhedron steadily "grows" by one or more extreme points at a time until $\mathcal{P}$ is completely generated.

The only remaining issue is the initialization. The application of linear program **LP3** requires that there be at least $m + 1$ affinely independent columns, which should be elements of the frame, and which contain the origin in their convex hull. Otherwise the linear program is not guaranteed to be feasible for any right-hand side and the powerful inferences possible from Result 2 would be invalid. There are several ways to assure that these initialization conditions are attained. We propose that the procedure be initialized in the following manner. Find the vector in $\mathcal{A}$ with greatest norm. This point is necessarily an element of $\mathcal{F}$ (see Result 2 in [4]). Take the negative of this "max-norm" vector and use it as the right-hand side element of the linear program **LP2**. The resultant basic feasible solution, *if unique*, is composed of m more elements of the frame from Result 3. (If not unique select another right-hand side which is the negative of some other element of the frame until one is found which generates a unique optimum.) These m vectors contain the right-hand side in their positive cone; therefore, applying Farkas' Lemma we may conclude that the m vectors in conjunction with the negative of the right-hand side vector constitute an affinely independent set of $m + 1$ vectors that positively span the space. Moreover, the convex hull of these vectors necessarily contain the origin (apply Stiemke's Theorem of Alternative). Note that this initialization scheme essentially identifies $m + 1$ points from the frame of $\mathcal{A}$, the convex hull of which is an m-dimensional simplex which contains the origin in its interior. Also, by selecting the "max-norm" vector as the "seed" for the right-hand side of **LP2** we may suppose that the resultant simplex is, in some sense, large (for more on how theorems of alternatives play a role in these ideas and on how this initialization scheme generates a "large" simplex, see [2]).

Notice that procedure **LPFindFrame** based on the linear program formulation **LP3** is fundamentally different from the standard LP-based approach. Here we "build-up" the polytope. The procedure using **LP3** generates linear programs that grow by one column every time a new vertex of $\mathcal{P}$ is identified. In the case of **LP1** the size of the linear program starts at m by $n - 1$ and, if enhancements are implemented, the number of columns may be reduced by removing points that are discovered not to belong to the frame. Since the columns used in **LP3** are always elements of the frame, the size of the final linear program is determined by the total number of extreme points of $\mathcal{P}$ and the size of each intermediate linear program is the total number of extreme points of $\mathcal{P}^E$. On the other hand, a difference which favors the approach based on **LP1** is the necessity of calculating and comparing inner product values in Steps 4 and 5. From our computational results in [3] we conclude that this difference is not enough to offset the advantages of the new procedure.

An important concern in the new method is the complication that arises from the presence of ties in Steps 4 or 5. Ties among three or more points are resolved by finding the frame of the points participating in the tie. However, finding just one element of this nested frame problem is sufficient to be able to proceed. A simple sorting as in "Preprocessor 1" of [4] will yield such a point. The inclusion of a point in $\mathcal{A}^E$ means that the current polytope changes its shape.

6 Parallel Formulation

We wish to consider the implementation of the new LP-based approach in a parallel MIMD environment, the SEQUENT SYMMETRY S81 with 20 processors, each equipped with a Weitek 1167 floating-point accelerator. As with the serial implementation we made use of the XMP linear programming code written by Roy Marsten for the solution of LP problems. We intended that the XMP code be modified to run multiple LP problems concurrently. We had two primary goals in mind in designing the modified parallel XMP code: (1) all LP problems should share the basic problem data, avoiding wasteful duplication and inefficient use of the memory resource, and (2) modify as little of XMP as possible. Note that XMP makes use of an array **STATUS** with length the number of problem variables in two ways: (1) entry j may contain zero or one of several negative numbers which indicate that variable j is nonbasic at upper or lower bound or is fixed at one of those bounds, or (2) entry j may contain a positive number which is the row in which variable j pivots and thus also indicates that variable j is basic. This array will have to be replicated for each concurrent LP problem because of usage (2). This dual usage also has an effect on the amount of synchronization needed in the overall implementation.

Consider the following procedure which combines features of both **ProcessPoint** and **LPFindFrame**:

Begin Procedure **Process**(a^k)

Step 1. If $a^k \notin \mathcal{A}^U$ exit the procedure.

Step 2. Determine if a^k belongs to $\mathcal{P}^E$ by solving **LP3**. If so, remove a^k from $\mathcal{A}^U$, add a^k to $\mathcal{A}^N$, and exit the procedure.

Step 3. Generate the direction $v \in \Re^m$ normal to a hyperplane separating a^k and $\mathcal{P}^E$, by setting $v = \pi^*$, the optimal dual solution to **LP3**.

Step 4. Calculate the maximum of the inner products $\langle v, a^p \rangle; \forall a^p \notin \mathcal{A}^N$. Let $\mathcal{A}^{max}$ be the set of all points $a^p \notin \mathcal{A}^N$ which attain this maximum. Identify one or more extreme points of the set $\mathcal{A}^{max}$ itself. Remove all such identified points which are also from $\mathcal{A}^U$ and add them to $\mathcal{A}^E$.

Step 5. Calculate the minimum of the inner products $\langle v, a^p \rangle; \forall a^p \notin \mathcal{A}^N$. Let $\mathcal{A}^{min}$ be the set of all points $a^p \notin \mathcal{A}^N$ which attain this minimum. Identify one or more extreme points of the set $\mathcal{A}^{min}$ itself. Remove all

such identified points which are also from $\mathcal{A}^U$ and add them to $\mathcal{A}^E$.
Step 6. Return to Step 2.

End Procedure

Note that expression "$a^k \notin \mathcal{A}^N$" in **Process(a^k)** has replaced expressions "$a^p \in \mathcal{A}^U$" and "$a^p \in \mathcal{A}^U \cup \mathcal{A}^E$" in Steps 4 and 5, respectively, of **LPFindFrame**. Also in Step 4 of **Process(a^k)** we "remove all such identified points *which are also* from $\mathcal{A}^U$ and add them to $\mathcal{A}^E$", whereas in the same step of **LPFindFrame** we only "remove all such identified points from $\mathcal{A}^U$ and add them to $\mathcal{A}^E$". ¿From the definitions, $a^k \notin \mathcal{A}^N$ and $a^p \in \mathcal{A}^U \cup \mathcal{A}^E$ are equivalent and $a^k \notin \mathcal{A}^N$ includes not only points $a^p \in \mathcal{A}^U$, but also points $a^p \in \mathcal{A}^E$ at which the maximum would not occur. Apparently Step 5 remains the same in both procedures and Step 4 of **Process(a^k)** has added some inefficiency and redundancy. We shall see that in the implementation environment, these changes are necessary.

The set of procedures $\{$ **Process(a^k)** $: k = 1, \ldots, n\}$ constitute a partitioning of the frame problem into n problems which are essentially independent in that no knowledge of the results of any other problem solution is necessary to carry out any other problem solution. Hence the frame problem is a good candidate to be cast as an asynchronous parallel algorithm in the sense of Part 2 of [1]. This could in fact be achieved, except for the dual use of the array **STATUS** by XMP.

The code uses a startup serial portion which sets up the data mappings for later concurrent use by the processors and finds an initial basis exactly as the serial implementation so that we begin with $m+1$ known extreme points in $\mathcal{F}$. The only modification actually necessary to the XMP system was to change the routine XMAPS which maps all the data structures into two large arrays MAPI and MAPR containing integer and real data, respectively. The mappings were modified to allow several processors to access the problem data and to allow separate basis inverse handling. The initialization within XMAPS of a labeled common block used by the routines which handle the basis was also moved to our interface routines which call XMP routines.

We also had to determine which working arrays needed to be replicated for local data and which contained fixed information which could be placed in shared memory. All this was handled in our interface routines and was thus transparent to the XMP system itself.

We used a single point-length shared-memory array **STATPT** to hold the current status of all points, using the following scheme:

(1) **STATPT**(k) = -1 *if and only if* $a^k \in \mathcal{A}^N$.
(2) **STATPT**(k) = 0 *if and only if* $a^k \in \mathcal{A}^U$.
(3) **STATPT**(k) = 1 *if and only if* $a^k \in \mathcal{A}^E$.

After the starting basis was obtained all the additional processors were forked off from the parent processor so that all processors had identical copies of the same advanced basis. Each processor then proceeded in two phases. In the first phase points were selected from $\mathcal{A}$ for processing in an implementation of **Process**(a^k) in such a way that all processors partitioned $\mathcal{A}$ without overlap. Each processor actually looked for points to process by stepping through array STATPT in steps the size of the number of processors with starting position offset by the processor number. Any processor finishing the first phase then enters the second or "mopup" phase. Any processor in the mopup phase looks for work by moving sequentially through STATPT in the *opposite* direction to that used in the first phase. Thus early finishers in the first phase "poach" work from slower finishers allowing all processors to finish work at about the same time. No synchronization was used in phase two, so that it is possible more than one processor in phase two could be working concurrently on the same problem or on a problem being worked on by a processor in the first phase. It was felt that the results were good enough and the resulting implementation was simple enough to justify not introducing extra synchronization overhead.

As the processors work, the underlying XMP problem data reflects the current status of $\mathcal{A}^E$, $\mathcal{A}^U$, and $\mathcal{A}^N$, as given by STATPT. All points in $\mathcal{A}^U$ and $\mathcal{A}^N$ have their entry in STATUS set at -4, indicating that the corresponding variable is fixed at a lower bound of 0. All points in $\mathcal{A}^E$ and $\mathcal{A}^N$ have their entry in STATUS set at a nonnegative value, either 0 indicating the variable is nonbasic at 0 or a positive value indicating the variable is basic and specifying the row in which it pivots. Thus when a point is removed from $\mathcal{A}^U$ and placed in $\mathcal{A}^E$, the corresponding entry in STATUS needs to be reset to 0, which then allows it to enter the basis when appropriate.

As processors work on specific points they need to post their determinations of point status. Essentially all that would be necessary would be to enter a 1 or −1 in the appropriate position in STATPT and reset the appropriate position in STATUS to 0 in every processor's local copy. But a complication arises because of the dual use of the array STATUS by XMP. If all processors were allowed to perform the updates above without synchronization a "race condition" exists. A processor could notice that a variable just changed its STATUS entry to 0 and pivot it into the basis, whereupon another processor posting the same variable as extreme resets the STATUS entry in all processors to 0 with disastrous consequences for the processor which had just pivoted.

Thus the posting must be handled carefully and only the first processor posting a variable as extreme can be allowed to change the appropriate STATUS entry for all processors.

The acual DYNIX FORTRAN code used for this synchroniztion is given below. Examination of the code shows that so few operations must must be carried out in a "locked" state that we feel justified in characterizing this parallel algorithm as "nearly" asychronous. In fact, if we were to use two arrays in place of STATUS, one for fixed variable status or a new "eligible to enter the basis" status and one for nonbasic at a bound status or basic and pivoting in a specific row, the algorithm could become truely asychronous. However, this violates our stated goal of keeping modifications of XMP to a minimum as this would require extensive recoding of XMP.

```
      SUBROUTINE CRIT(PT,ST,PTS,PPN,STATUS,KNOWN,FXCOLS)
      INTEGER PT,ST,PTS,PPN,STATUS(PTS,PPN),KNOWN,FXCOLS,I
C  expect ST = 1 if point PT is identified as extreme
C  expect ST =-1 if point PT is identified as nonextreme
C  PTS    = number of points
C  PPN    = number of processors
C  KNOWN  = count of points whose status is known
C  FXCOLS = count of points whose LP columns are fixed at 0
      IF(STATPT(PT).NE.0) RETURN
      CALL m_lock
        KNOWN              = KNOWN+1
        STATPT(PT)         = ST
        IF(ST.GT.0) THEN
          FXCOLS           = FXCOLS-1
          DO 10 I=1,PPN
          STATUS(PT,I) = 0
10      CONTINUE
      ENDIF
    CALL m_unlock
    RETURN
    END
```

Another potential race condition was avoided by the modifications to Steps 4 and 5 previously noted for **Process**(a^k). While a given processor is computing the indicated inner product maximum another processor could post a point as extreme that was not extreme when the given processor finished its LP problem. If the point in question was where the maximum would have occurred relative to the view of $\mathcal{A}^U$ present when Step 2 finished, a false extreme point detection could result. The additional modification to Step 4 is necessary for similar reasons. In the actual coding of Steps 4 and 5, it is most efficient to compute the maximum and minimum over the same points while computing the inner products once.

The only question which remains is whether or not this algorithm could somehow cycle in the sense that the processors operating concurrently keep reidentifying the same points as extreme that have already been identified as extreme and fail to identify any new points as nonextreme.

Result 8. *If $\mathcal{A}^U$ in not empty and at least two processors are working on two distinct points executing* **Process**(a^k)*, then at least one point of $\mathcal{A}^U$ will be identified as extreme or nonextreme.*

Suppose to the contrary, that no new points are identified. Then $\mathcal{A}^E$, $\mathcal{A}^U$, and $\mathcal{A}^N$ do not change. Hence each instance of **Process**(a^k) being executed is an instance of procedure **ProcessPoint**, which implies that for every executing processor at least one point leaves $\mathcal{A}^U$ in either Step 2 or Step 4.

7 Computational Results

For test problems we have used the same three moderate-sized DEA problems employed in the testing of the new LP-based approach (see [3]). This data was preprocessed, ordering it by distance from the origin. In so doing it is likely that those furthest from the origin will be extreme points (the furthest is guaranteed to be extreme). Characteristics of these problems are given in Table 1 below along with the best serial time reported in [3]. The serial time will be used in computing speedup. All times are wall clock times in seconds on the Sequent Symmetry S81.

Table 1. Test Problems

Prob. No.	Data Characterstics				
	Source	Points	Dimension	Extreme Point Density	New LP Serial Time
1	Actual	334	19	100%	993
2	Actual	816	14	90%	1890
3	Generated	1000	7	34%	651

Table 2. Average Run Time and Speedup Factors

No. of Procs	Average Run Time			Speedup		
	334 pt. prb.	816 pt. prb.	1000 pt. prb.	334 pt. prb.	816 pt. prb.	1000 pt. prb.
1	1264	2499	785	0.79	0.76	0.83
2	593	1231	432	1.68	1.54	1.51
3	426	848	314	2.33	2.23	2.07
4	331	640	234	3.00	2.95	2.79
5	272	528	207	3.65	3.58	3.14
6	238	449	160	4.18	4.21	4.07
7	212	407	156	4.68	4.64	4.16
8	192	371	148	5.16	5.10	4.39
9	174	335	130	5.70	5.65	5.01
10	165	307	122	6.03	6.16	5.32
11	154	283	114	6.45	6.68	5.69
12	147	268	107	6.76	7.06	6.07
13	143	266	98	6.93	7.11	6.67
14	133	251	95	7.47	7.52	6.85
15	131	227	91	7.56	8.33	7.18
16	132	240	89	7.50	7.88	7.31

The test runs employed a fixed number of processors varying from 1 to 16 processors executing the parallel code. These runs were not made in a dedicated environment, and are thus subject to the varying influence of the rest of the system load. Since the serial times reported in [3] included data input time, we have chosen to include this in our timings. Hence our speedup factors may be considered to be conservative. The test runs consisted of three repetitions of each of the three problems for each fixed number of processors. The average run times and speedup factors are given in Table 2.

Figures 1-3 show plots of the run times and speedup factors for each problem. The run time plots include the maximum and minimum run times to give an idea of the variability.

$\boxed{\text{Figure 1 about here}}$

$\boxed{\text{Figure 2 about here}}$

$\boxed{\text{Figure 3 about here}}$

As was expected, the times for one processor are inferior to the (one-processor) serial times. This is due in part to the increased overhead of the parallel code, but mostly due to the fact that the parallel code is not "smart" in computing inner products and cannot bypass the ones corresponding to points already declared as extreme in Step 4 and cannot abandon further inner products when an exteme points gives a smaller inner product value than the minimum inner product over the unknown status points in Step 5. The speedups achieved appear to be good, leveling off at 14 to 16 processors while achieving speedups of 7 to 8 in that area.

Figure 4 shows how the work is distributed among the processors and Figure 5 shows that the magnitude of the major components of the work does not vary greatly for an average LP solution over the range of the processors.

$\boxed{\text{Figure 4 about here}}$

$\boxed{\text{Figure 5 about here}}$

8 Concluding Remarks

The primary motivation for this research has been to provide a resource for large scale applications which require finding the frame of the convex hull.

Applications equivalent to the frame problem in data envelopment analysis routinely exceed $n = 8,000$ to $n = 10,000$ points over fewer than $m = 20$ dimensions. In these applications, it takes several hours to identify the frame applying the conventional methods of solving n linear programs with m rows and $n - 1$ columns. Similar situations exist in stochastic programming. In general, the state-of-the-art in techniques for finding the frame is such that the methodology limits the size of the applications can be addressed. As Wallace and Wets [7] state: "there is a lot to be gained by a more efficient implementation (of an algorithm to find the frame of the convex hull, than one based on solving L.P.'s)". Our investigations on parallelizing our new procedure based on solving linear programs which begin small and increase progressively in size have shown that we can realistically expect a reduction equivalent to one order of magnitude in the solution times. These developments will permit the solution of problems previously considered too large.

References

[1] Bertsekas, D.P., and J.N. Tsitsiklis, *Parallel and Distributed Computation*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1989.

[2] Dulá, J.H., "Designing a majorization scheme for the recourse function of two-stage stochastic linear programs," *Computational Optimization and Applications*, Vol. 1, No. 4, 1993.

[3] Dulá, J.H. and R.V. Helgason, "A New Procedure for Identifying the Frame of the Convex Hull of a Finite Collection of Points in Multidimensional Space," Tech. Report 93-1, Southern Methodist University, Dallas, Texas, 1993.

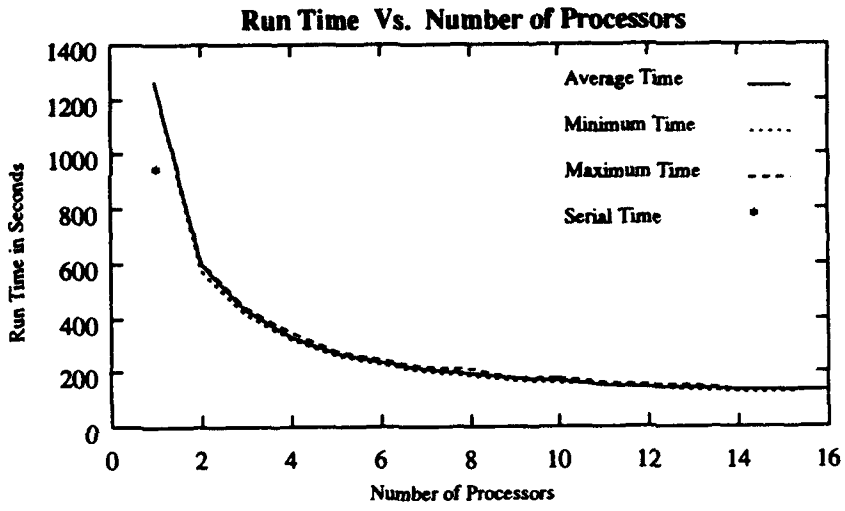
[4] Dulá, J.H., R.V. Helgason, and B.L. Hickman, "Preprocessing schemes and a solution method for the convex hull problem in multidimensional space," in *Computer Science and Operations Research: New Developments in their Interfaces,* O. Balci, ed., Pergamon Press, UK, 1992.

[5] Rosen, J.B., G.L. Xue, and A.T. Phillips, "Efficient computation of extreme points of convex hulls in $\Re^d$," Preprint 91-42, Army High Performance Computing Research Center, University of Minnesota, Minneapolis, MN 55415, 1991.

[6] Wallace, S.W. and R.J.-B. Wets, "Preprocessing in stochastic programming: the case of uncapacitated networks," *ORSA Journal on Computing,* Vol. 1, No. 4, pp. 252-270, 1989.

[7] Wallace, S.W. and R.J.-B. Wets, "Preprocessing in stochastic programming: the case of linear programs," *ORSA Journal on Computing,* Vol. 4, pp. 45-59, 1992.

[8] Wets, R.J.B. and C. Witzgall, "Algorithms for frames and lineality spaces of cones," *Journal of Research of the National Bureau of Standards – B Mathematics and Mathematical Physics*, Vol. 71B, No.1, pp. 1-7, 1967.



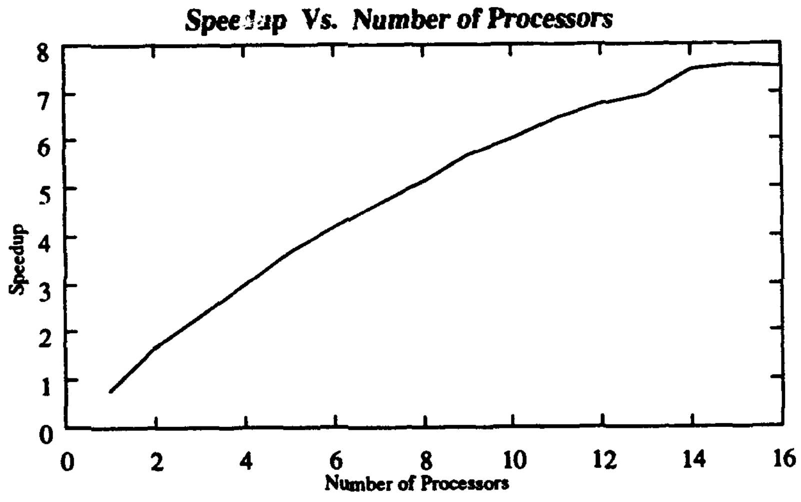
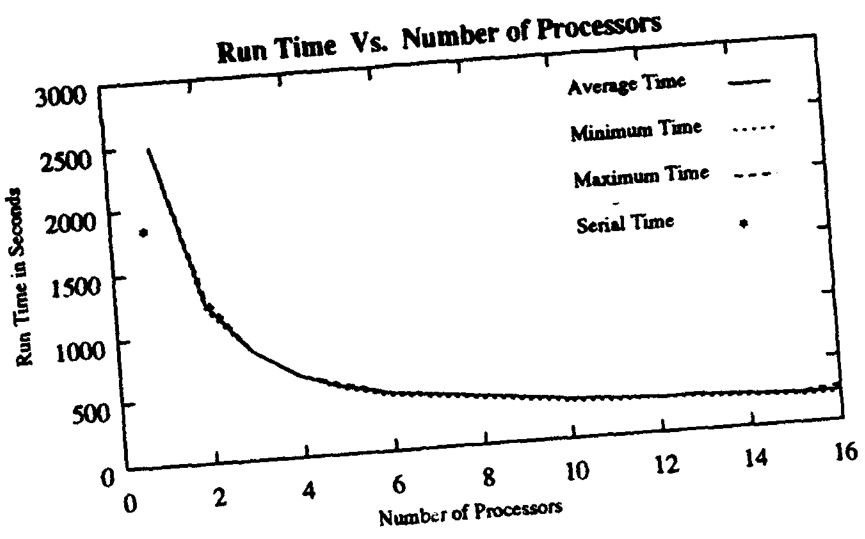


Figure 1: 334 Point Problem



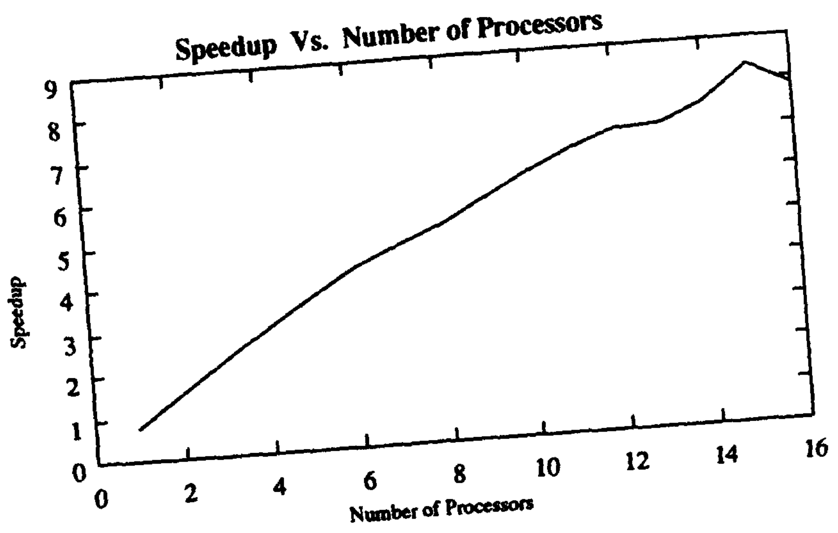
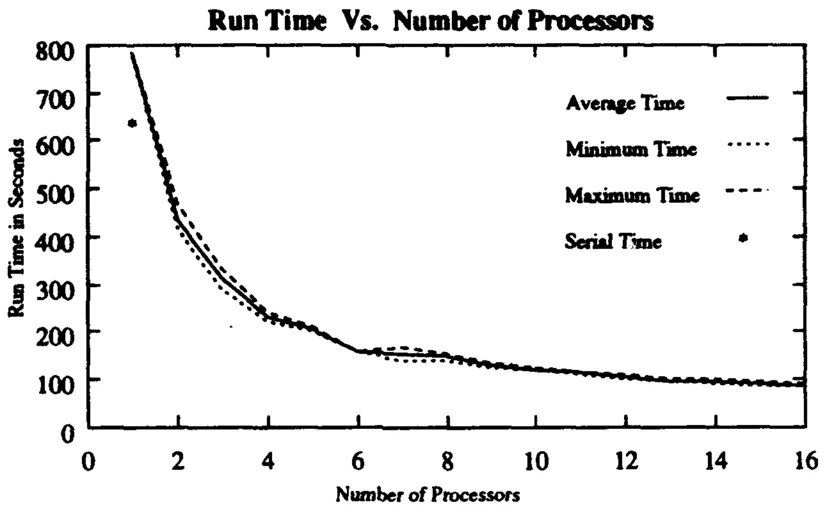


Figure 2: 816 Point Problem



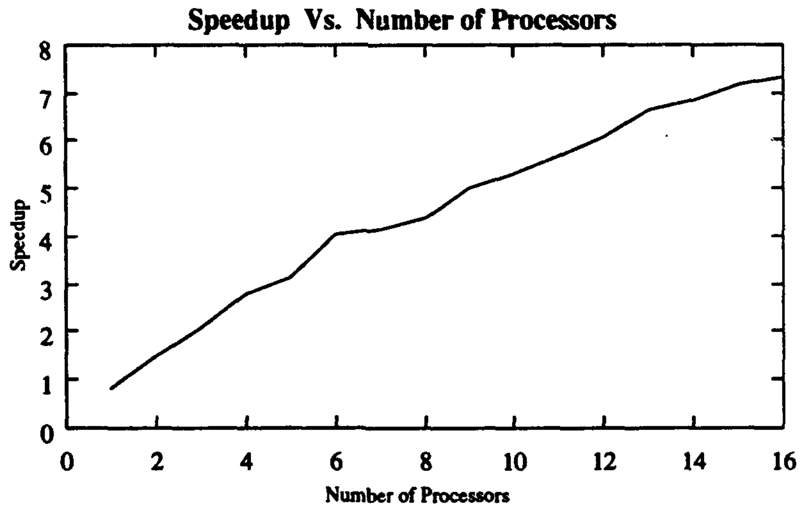
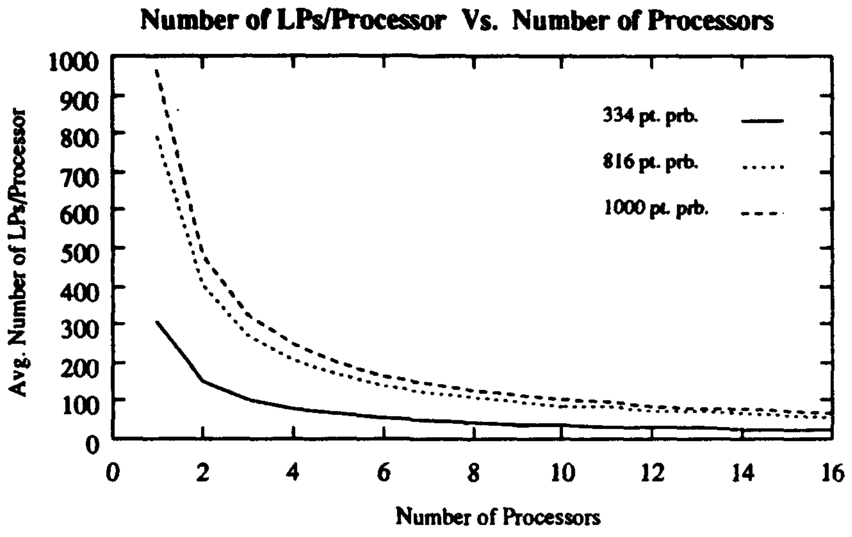


Figure 3: 1000 Point Problem



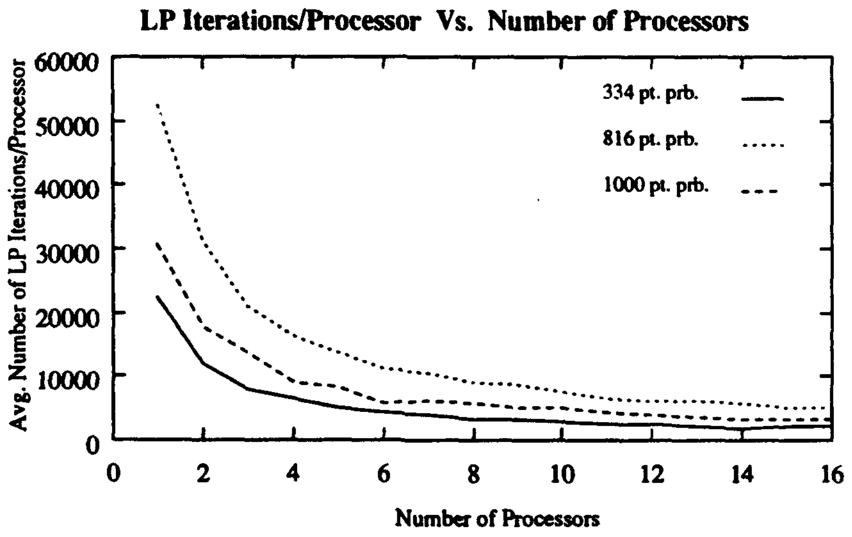
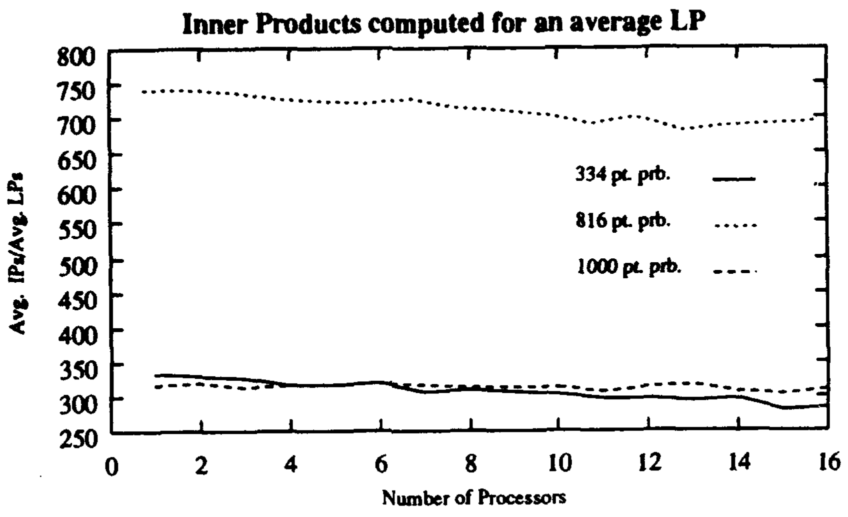


Figure 4: Work Distribution amongst the processors



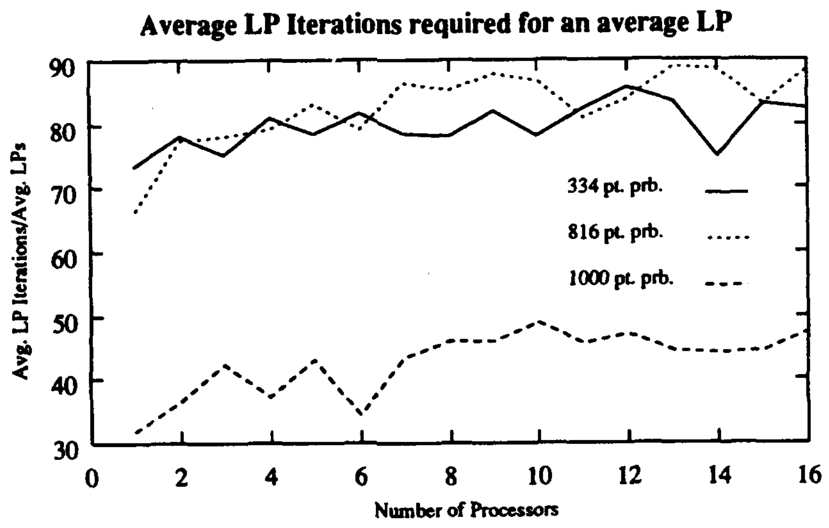


Figure 5: Work distribution per solved LP

Distribution List

Stanley C. Collyer, Chairman
Manpower Research and Development Planning Committee
ONT 222
800 North Quincy Street
Arlington, VA 22217-5000

Donald Wagner
Code 1111 MA
Office of Naval Research
800 Quincy Street
Arlington, VA 22217-5000

Thomas A. Blanco, Head
Assignment Systems Division
Navy Personnel Research and Development Center
Code 112
San Diego, CA 92152-6800

Timothy Liang
Navy Personnel Research and Development Center
Code 112
San Diego, CA 92152-6800

Iosef Krass
Navy Personnel Research and Development Center
Code 112
San Diego, CA 92152-6800

Wally Sinaiko
Manpower Research and Advisory Services
Smithsonian Institution
801 N. Pitt Street #120
Alexandria, VA 22314-1713

Carole Voltner, Assistant Director
Office of Research Administration
SMU
Dallas, TX 75275